University of Glasgow

COMPUTING SCIENCE DEPARTMENT

PASCAL COMPILER

CHAPTER 5 : CODE GENERATION

Internal Document Only

Assembly

John Cavouras.

UNIVERSITY OF GLASGOW

Computing Department

PASCAL COMPILER

CHAPTER 5 : CODE GENERATION

Internal Document Only

Assembly

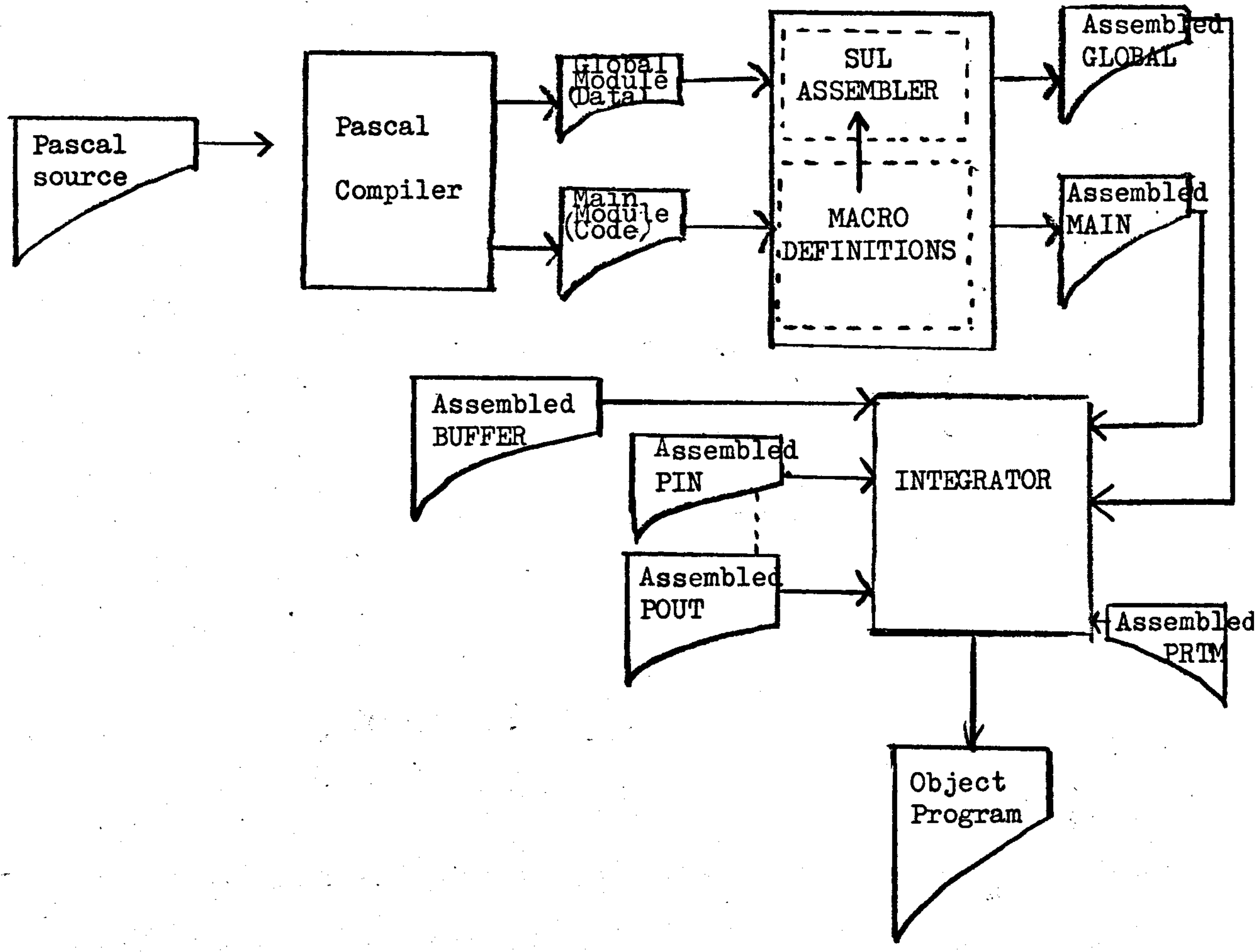
John Cavouras.

July 1973.

5. CODE GENERATION

The first stage of the code generation by the Pascal compiler is a series of SUL (symbolic usercode language) macro calls. These macro calls are expanded by the macro processing part of the assembler and after the second pass relocatable code is produced.

The third stage is the linking of this code with the standard system subroutines (i.e. buffering and conversion BUFFER, I/O modules PIN, POUT and run-time monitor PRTM) by the integrator. This produces a relocatable binary program capable of being run on an ALP2/3 processor



Section 5.1 deals with the Macro definitions and Section 5.2 with the run-time monitor and the integration.

5.1 /

5.1 CODE GENERATION MACROS

In this section the macro definitions are given. Macros denoted by [1] are generalised, i.e. could be usefully used by programs other than the ones produced by the Pascal compiler. On the other hand, macros denoted by [2] should not be macros at all. They either generate one element or have no conditional processing.

Each macro name and parameters form the loading for that macro. The parameters are then explained and their allowable values are given. Finally a flow chart of the macro body is illustrated.

INDEX

5.1,6	Repetitive commands	
	PINC	increment by one
	PDEC	decrement by one
	PFOR	for statement
5.1.7	Accessing array elements	
	PPP3	
	PIND	access an array element when index is not a subscripted variable
	POND	access an array element when the index is a subscripted variable
	PCHR	compare two character (packed) array elements
5.1.8	I/O	
	PINP	input
	POUT	output
	PTXT	output a character string
5.1.9	Run-time errors	
	PCHR	check at run-time
5.1.10	PRUB	heading macro

5.1.1 The basic macro used for manipulating scalar data is

PPPØ[1]=MNEM=REG=TYPE=OFFSET=LEVELS

1, 2. MNEM, REG: Any group 1 instructions mnemonic.

e.g. PPPO=ADD=A= . . . .

o r PPPO=STHS==

3. TYPE: Can be:

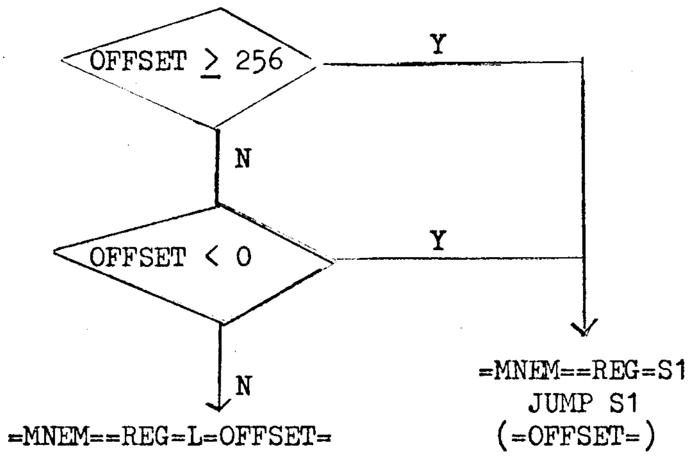
T	for a temporary storage location
I	for a literal
V	for a variable parameter
N	for a non local variable
G	for a global variable
L	for a local variable
C	for a constant parameter
X	for a non local parameter
Z	for an array element

4. OFFSET: An integer n such that $-128 \leq n < 128$

5. LEVELS: A non-negative integer,

'=TYPE='	Code generated
'L' or 'T' or 'C':	=MNEM==REG=P=OFFSET=

'I':



'V': =MNEM==REG=P=OFFSET=I

'G': =MNEM==REG=M(=LEVELS=)=OFFSET=

'N':

```
 LDRYP
 LOOP PØI } repeated =LEVELS= times
=MNEM==REG=P=OFFSET=
 EXRY P
```

'X': /

```
'X':        LDRY P
            LOOP PØI }    repeated = LEVELS=times
            =MNEM==REG=P=OFFSET=I
            EXRY P

'Z':        SETB P=LEVELS=
            LDRY B
            SETB P=OFFSET=
            +MNEM==REG=B Y
```

5.1.2 Fetch and Store Macros

These include fetch scalar PGET, store scalar PPUT, fetch or store vector element PABM and vector assignment PSTA.

PGET^2=REG=TYPE=OFFSET=LEVELS

Fetch a datum in register REG. 4 parameters as for PPPØ (section 5.1.1). A single element is generated:

```
PPPO=SET==REG===TYPE===OFFSET===LEVELS=
```

PPUT^2=REG=TYPE=OFFSET=LEVELS

Store the datum in REG at TYPE, OFFSET, LEVELS. 4 parameters as in PPPØ (section 5.1.1). The single element generated is

```
PPPO=ST=REG=S===TYPE===OFFSET===LEVELS=
```

PABM=OPER=REG=MHR1=MHR2

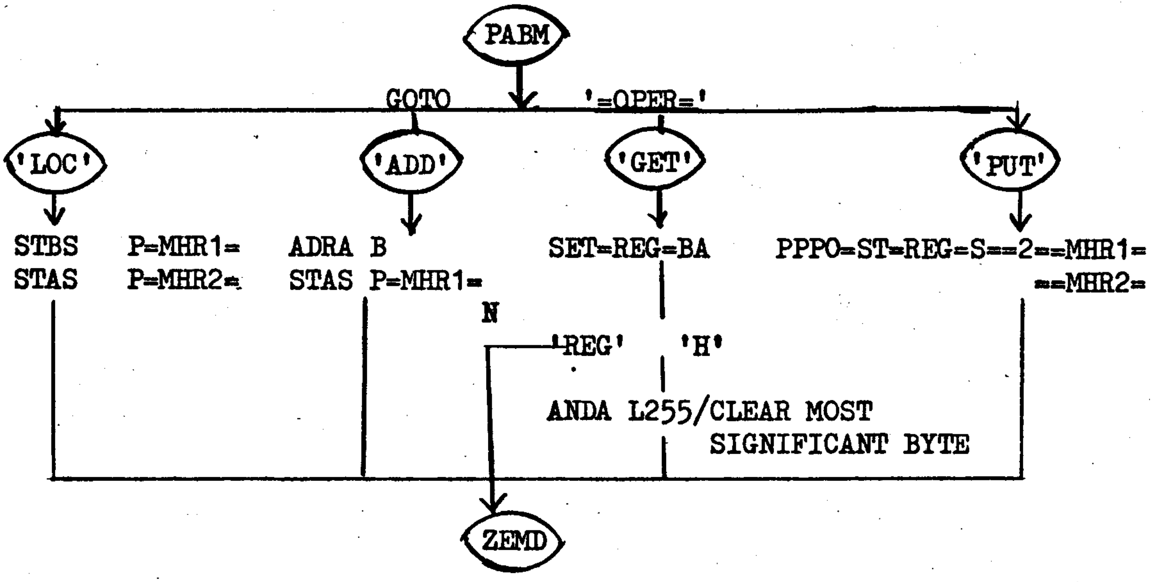
This is used for array element operations. It is neither integral nor general as PPPO (section 5.1.1). It cannot be used for double word arrays (version 1 does not implement them). 4 parameters.

1. OPER: PUT, ADD, LOC, GET

2. REG: A or B or H .

3. MHR1: A P relative address, containing the base address of the array for character arrays or the base address plus the index for single(or double) word arrays.

4. MHR2: A P relative address, containing the index for character arrays.



/

PSTA[1]=REG=GM1=GM2=LM1=LM2=SIZE=LA

Assign the subarray given in LM1, LM2 to the subarray given by GM1, GM2. By changing the elements numbered '*' below, it can be made general to deal with any array operation (PL/1 wise). 7 parameters. See also PEQA (section 5.1.4).

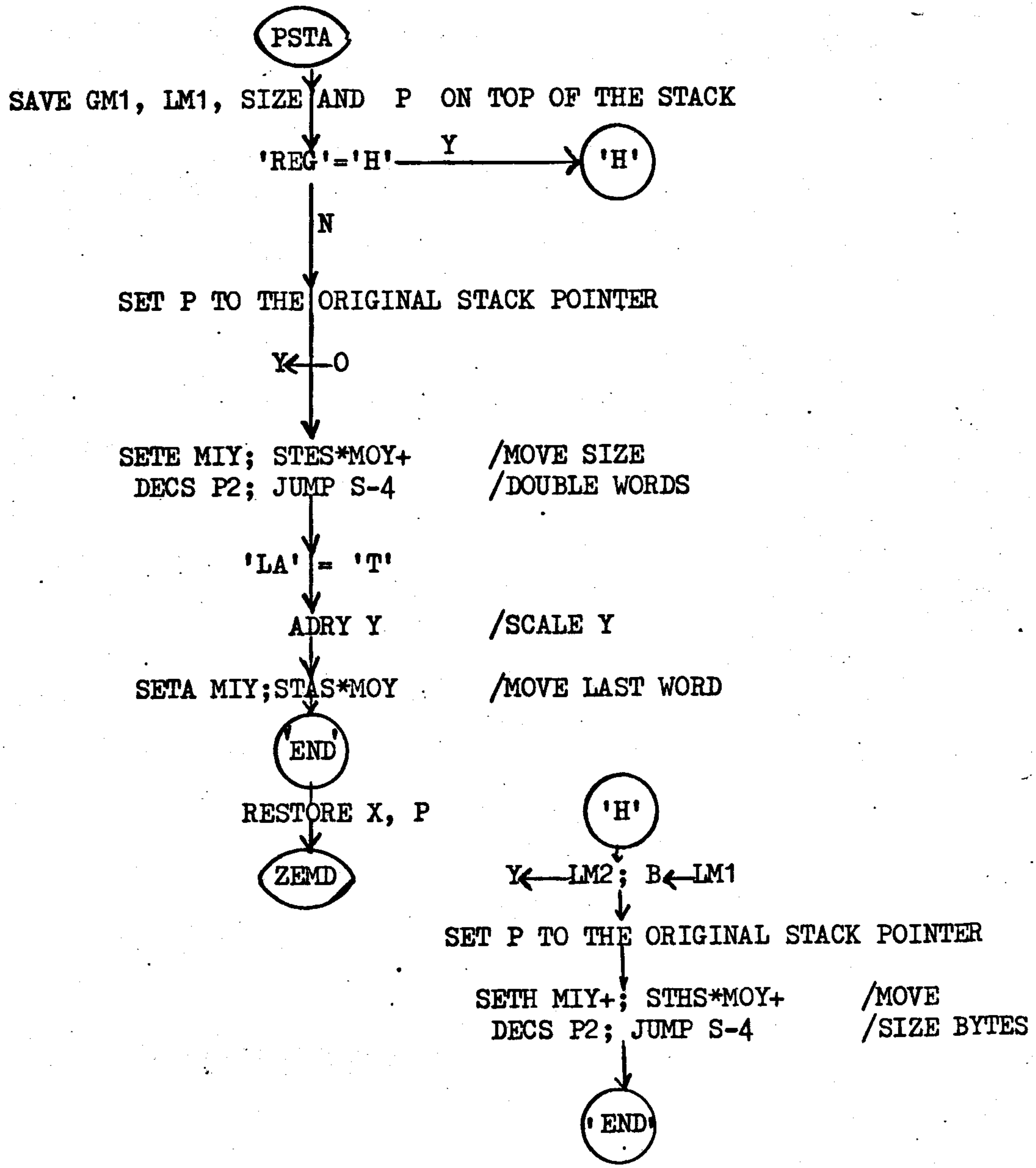
1. REG: A or B or E or H .

2,4 GM1,LM1: The base addresses of the arrays in the case of character arrays. The base addresses plus offsets of the arrays in the case of single or double word arrays.

3,5. GM2, LM2: The indices of character arrays.

6. SIZE: The number of double words or bytes to be assigned.

7. LA: T for odd size assignments of single word arrays.



Note: Single word arrays are treated as double (except for the last word if the number of elements is odd.) This is due to the Multum ability to perform automatic scaling when MR, or RR, instructions are executed.

PPP5[1]=REG=TYPE=OFFSET=LEVELS=LINENO=SRTN

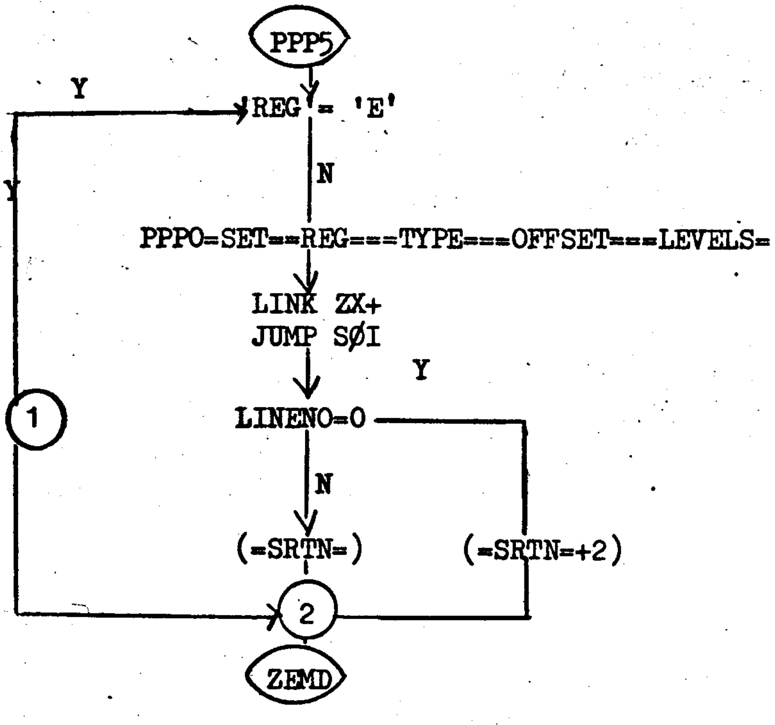
This is a generalised division macro. It was introduced so that the following four division macros, could be treated uniformly. It has 6 parameters. Provision for double length operands is similar to PMLT.

1. REG: A or E.

23,4: As in PPPO (section 5.1.1)

5: LINENO: The line number of the corresponding Pascal statement. If Ø no check for integer division by zero is performed.

6: SRTN: A subroutine name: DIVR for division, MODR for modulo, if check is required; otherwise DIVR+2, MODR+2.



Warning: If this macro is called with REG=E, no code will be generated.

PDIV[2]=REG=TYPE=OFFSET=LEVELS=LINENO

Divide REG by the datum in TYPE, OFFSET, LEVELS. 5 parameters as in PPP5. A single element is generated:

PPP5=B==TYPE===OFFSET===LEVELS===LINENO==DIVR

PMOD[2]=REG=TYPE=OFFSET=LEVELS=LINENO

Modulo. Same comments as PDIV. The element generated is

PPP5=B==TYPE===OFFSET===LEVELS===LINENO==DIVR.

/

PDVR^2=REG=TYPE=OFFSET=LEVELS=LINENO

Reverse division. Here two elements are generated:

```
LDR BA
PPP5=A==TYPE===OFFSET===LEVELS===LINENO=DIVR
```

PMDR^2=REG=TYPE=OFFSET=LEVELS=LINENO

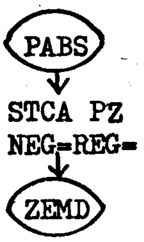
Reverse modulo. The elements generated are:

```
LDRBA
PPP5=A==TYPE===OFFSET===LEVELS===LINENO=DIVR
```

PABS^2=REG

Absolute value. One parameter

1. REG: A or E only



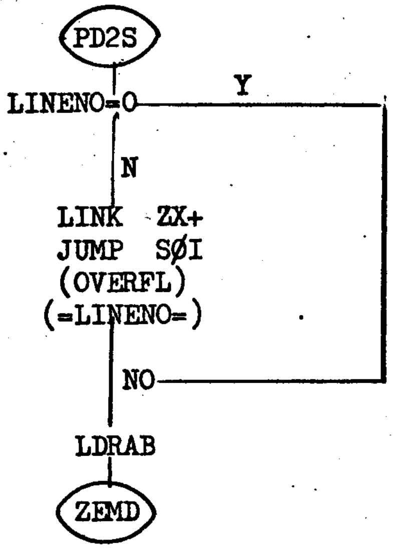
PNEG^2=REG

Negate A or E. The element generated is

```
NEG=REG=
```

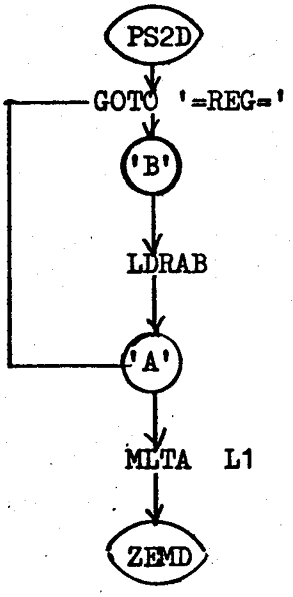
PD2S=LINENO

Contract from double length to single and check for overflow if LINENO is not zero.



PS2D^2=REG

Expand A or B to double length. Result occupies E. This macro is not called by the Version 1 compiler.



5.1.4 Flow of control macros

These include relations, boolean operations, selective flow of control including unconditional.

PJMP^{1}=LABEL=SKIP

This is a generalised GOTO statement.

1. LABEL: A positive unsigned integer.

2. SKIP: T if and only if the macro call follows a <compare/increment|decrement and skip> instruction.

If P=LABEL= is defined and is not more than 128 locations away, then

```
JUMP S (P=LABEL=-*) is generated.
```

otherwise if SKIP=T then

```
JUMP S1I
JUMP S1
(P=LABEL=)  are generated
```

otherwise if SKIP≠T then

```
JUMP SØI
(P=LABEL=)  are generated
```

PREL^{1}=CC=BOOL=LABEL

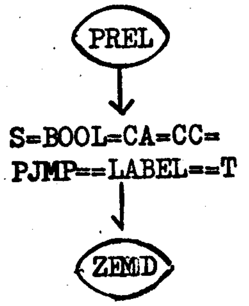
Relations code.

1. CC: ZZ, PZ, NN, PP, NZ, DD, IZ, DZ

   Condition for Group 3, format 2 instructions of the ABP2.

2. BOOL: T or F

3. LABEL: A positive unsigned integer.



PINS^{2}=LABEL

Generate/

Generate a label. A single element is generated

P=LABEL=

where LABEL is a positive unsigned integer.

PANS^2=LABEL

Generate a label available to other modules. A single element is generated:

#P=LABEL=

For the Pascal Compiler this label should be either MAIN or GLBL.

PONS^2=LABEL

Generate the address of a label. A single element is generated:

(P=LABEL=)

where LABEL is as in PINS.

PNOT^2=

Logical ¬ . A single element is generated.

NEVA L1.

PODD^2=

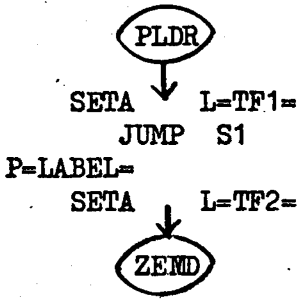
Are the contents of A odd? A single element is generated:

ANDA L1 .

PLDR=LABEL=TF1=TF2

Assigns the result of a relation to register A. TF1 is always ¬ TF2 .

1. LABEL:	A positive unsigned integer.
2. TF1:	T or F
3. TF2:	F or T .



```
     SETA   L=TF1=
        JUMP  S1
P=LABEL=
     SETA   L=TF2=
```

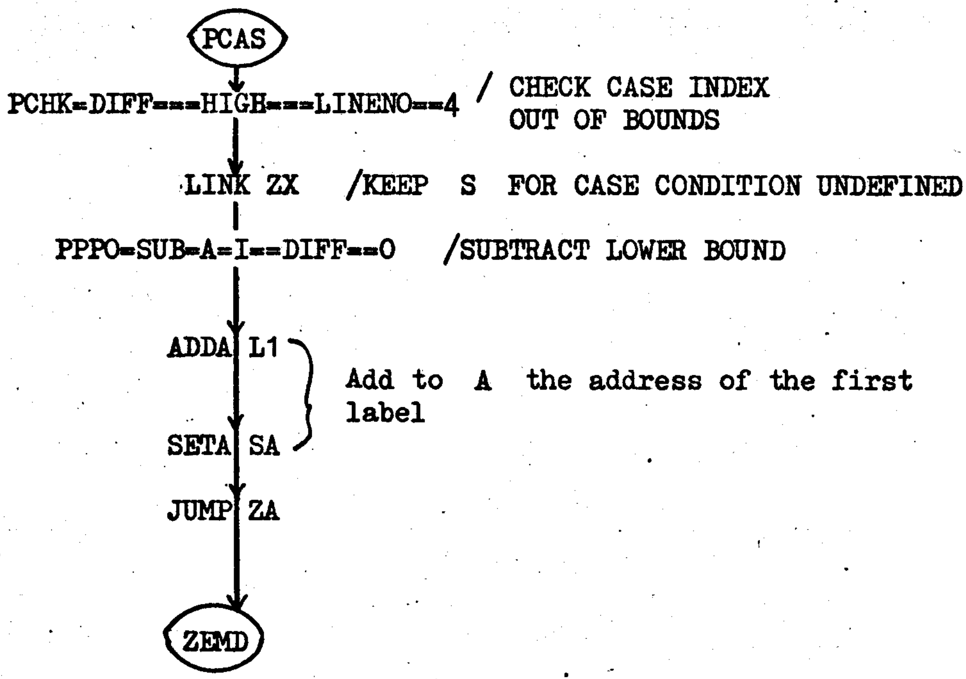
PCAS=DIFF=HIGH=LINENO

Case statement. 3 parameters.

1. DIFF: The lower bound.

2. HIGH: The upper bound.

3. LINENO: The line number of the corresponding Pascal source statement.


PCAS
PCHK=DIFF===HIGH===LINENO==4 / CHECK CASE INDEX OUT OF BOUNDS
LINK ZX /KEEP S FOR CASE CONDITION UNDEFINED
PPPO=SUB=A=I==DIFF==0 /SUBTRACT LOWER BOUND
ADDA L1
SETA SA
Add to A the address of the first label
JUMP ZA
ZEMD

PEQA=GM1=GM2=LM1=LM2=REG=SIZE=LA=BOOL

Test for equality between two arrays. 8 parameters. See PSTA section 5.1.2 .

1. GM1: if the array is a character array then its base address; if a word or double array then it is base address plus the offset of the element from which the comparison is needed.

2. GM2: the offset of the element onwards from which the comparison is needed for character arrays.

3. LM1: as GM1 for the second array.

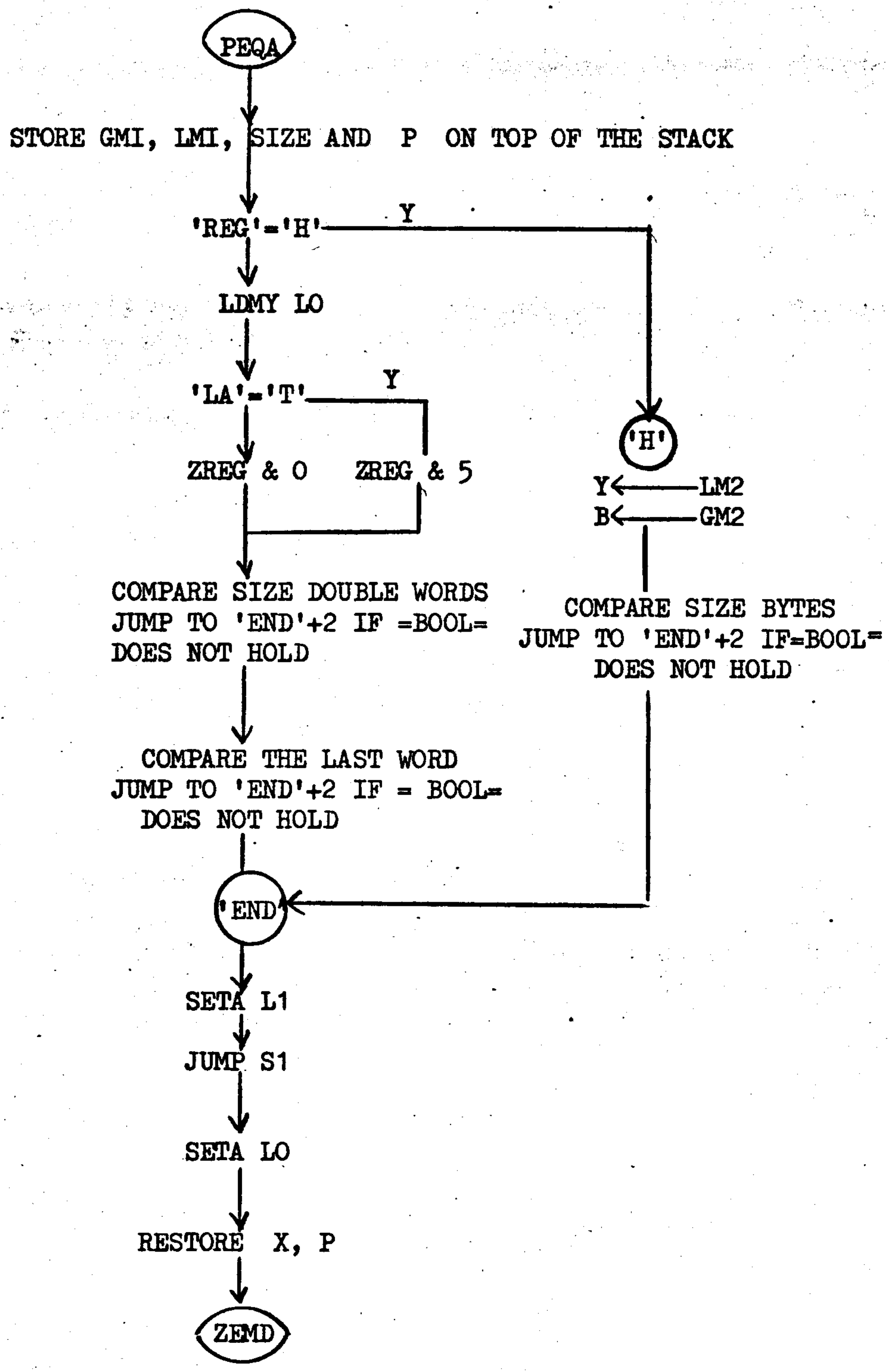
4. LM2: as GM2 for the second array.

5. REG: H for byte arrays.

6. SIZE: The number of double words or bytes to be compared.

7. LA: T for odd sized comparisons of single word arrays.

8. BOOL: T or F for testing equality of inequality.



Note: i.e. set A to one if the comparison is =BOOL= , otherwise to zero.
The assembler register & is used.

5.1.5 Stack Frame Macros

These include procedure call PCAL, set up the global pointers PGPS, procedure exit sequence PEXT, procedure entry sequence PENT, pass parameter values or addresses PXIN, fetch address of data PADR, initialise globals, and the storage allocation macros PPTR (static) and PALC (dynamic).

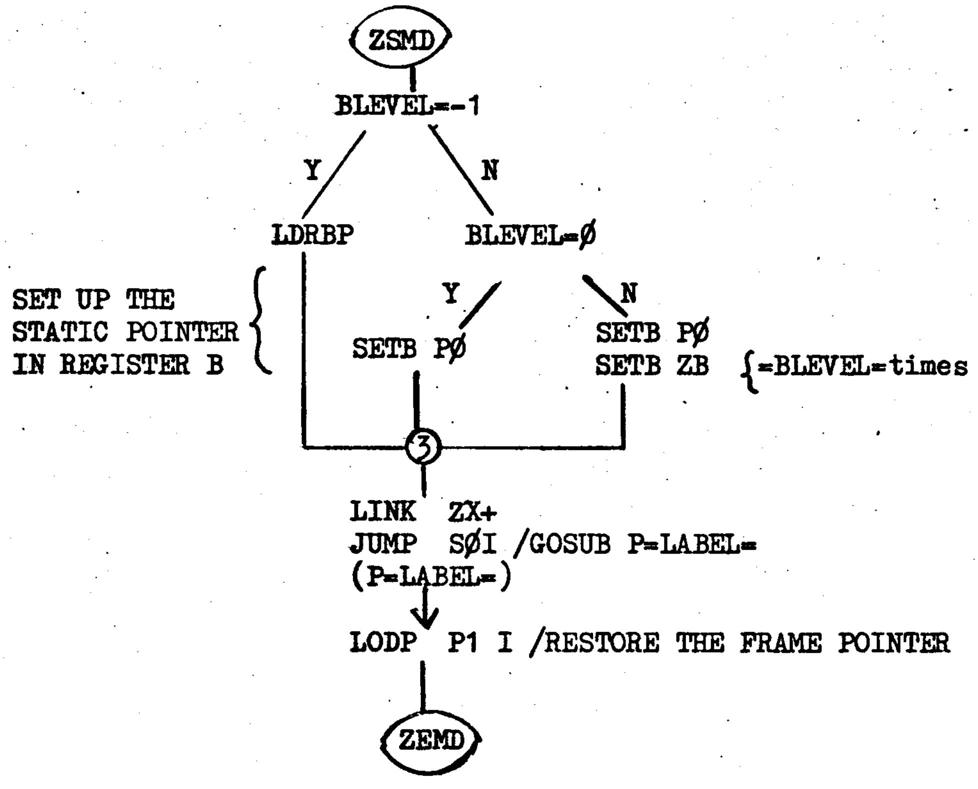
PCAL =BLEVEL=LABEL

Procedure Call. 2 parameters.

1. BLEVEL:

-1	:	The called procedure is local to the calling one
Ø	:	The called procedure is at the same level as the calling one
n (>0)	:	The called procedure is n levels out.

PARAMETER 2: LABEL: An unsigned positive integer.



PGPS=GLBL

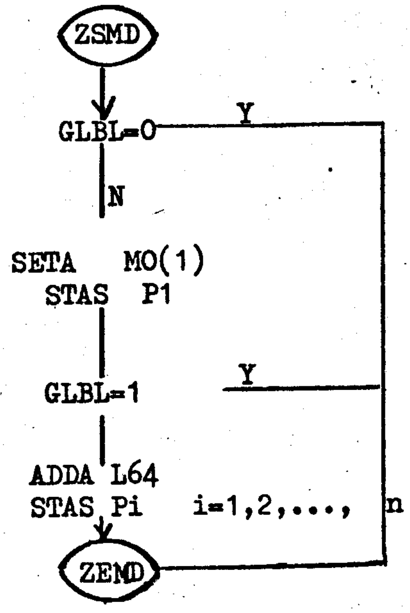
Set up the global pointers. 1 parameter

1. GLBL:

Ø	:	if no global pointers are needed (i.e. no global space or no globals accessed)
n	:	if n $\{ n \in I \mid 0<n<5 \}$ registers are required to span the global space.

Note: The Assembler register % is used.

/



PEXT[1]=PARS=SCAL1=FUNCT=REG

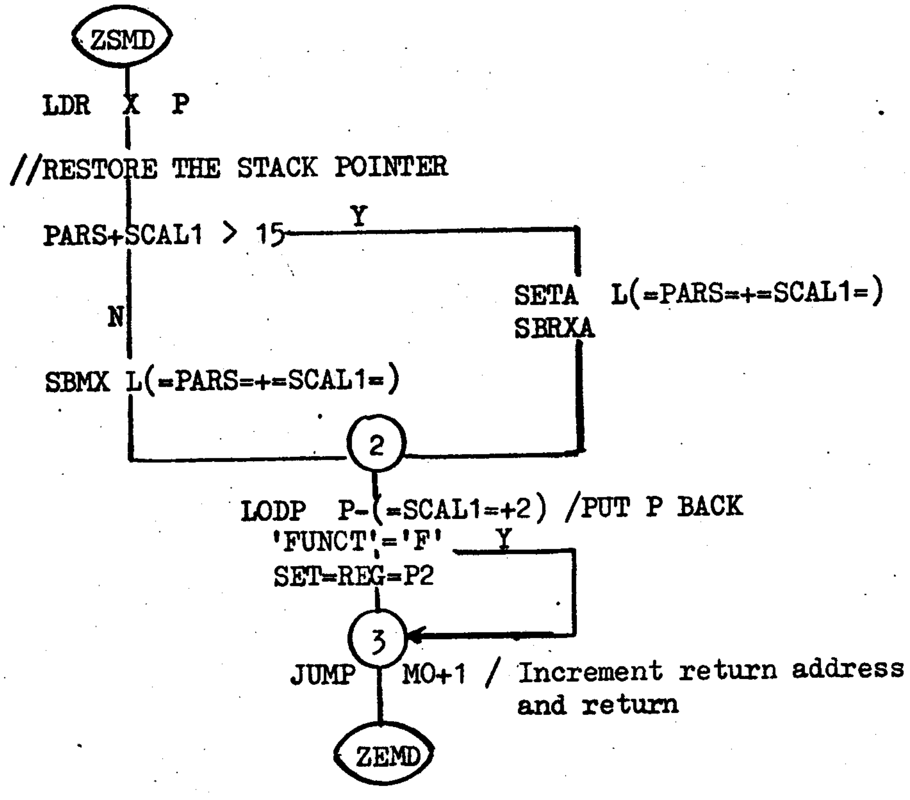
Exit sequence. 4 parameters.

1. PARS: The number of the procedure parameters plus 2. Two locations reserved for the return address and the dynamic pointer.

2. SCAL1: An integer ≥0

3. Funct: F if the procedure is not a function.

4. REG: A register name (A or E or B or H) if the procedure is a function -



PENT[1]=SCAL1=SCAL2=STRUCT=OMIT=LABEL

Procedure entry code. 5 parameters. A call of this macro should normally be followed by a call of the PGPS macro. The assembler register & is used.

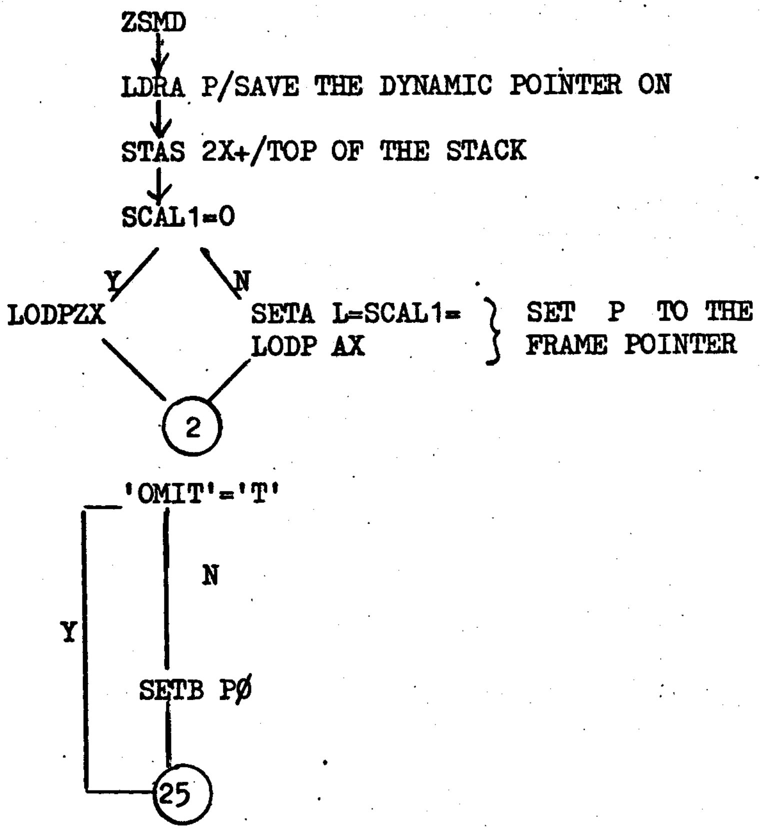
1. SCAL1: An integer ≥0

2. SCAL2: " " "

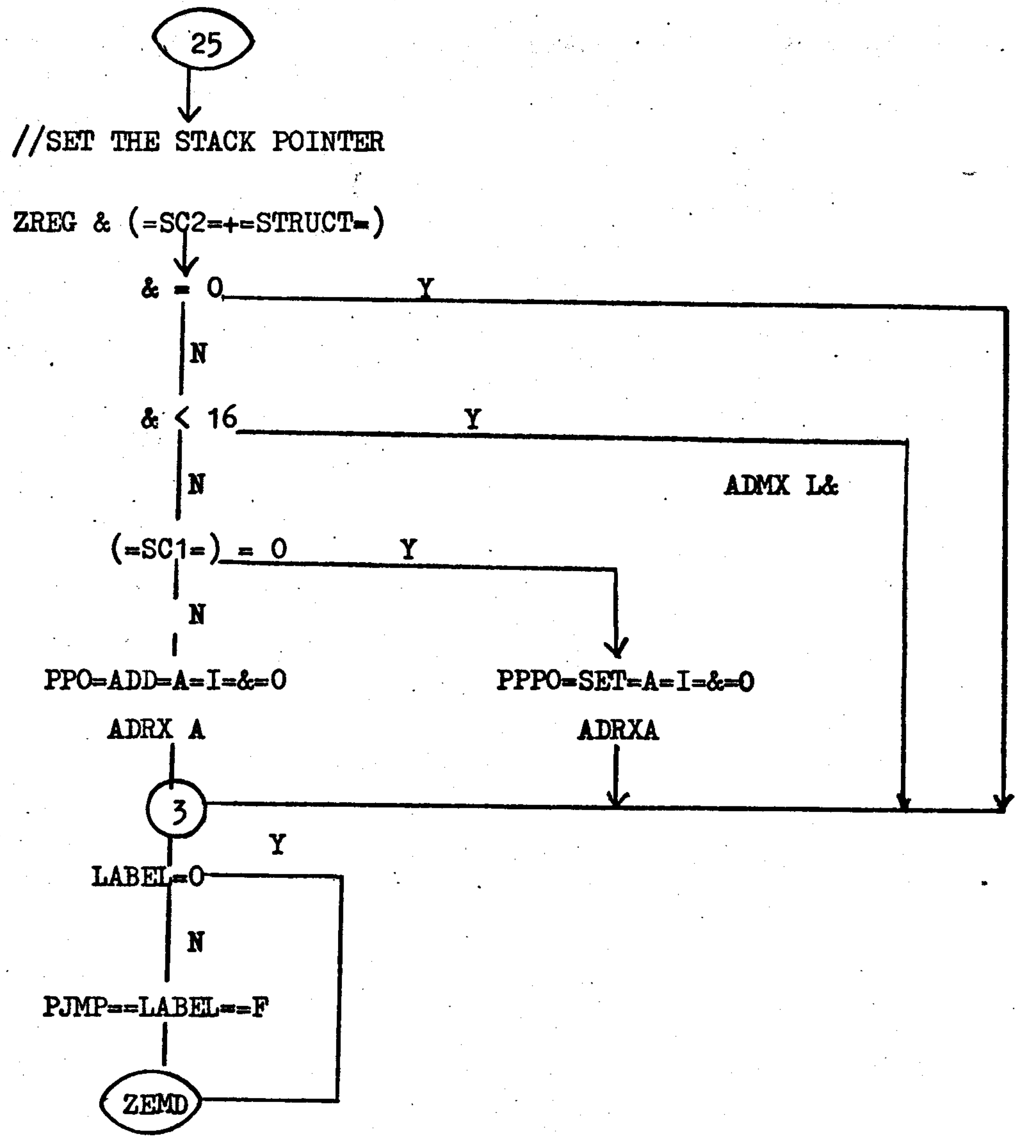
3. STRUCT: " " "

4. OMIT: T for not saving the static pointer (e.g. main program).

5. LABEL: Ø if the entry sequence is compiled last, otherwise an unsigned positive integer.



ZSMD
LDRA P/SAVE THE DYNAMIC POINTER ON
STAS 2X+/TOP OF THE STACK
SCAL1=0
Y
N
LODPZX
SETA L=SCAL1=
LODP AX
SET P TO THE FRAME POINTER
2
'OMIT'='T'
N
Y
SETB PØ
25

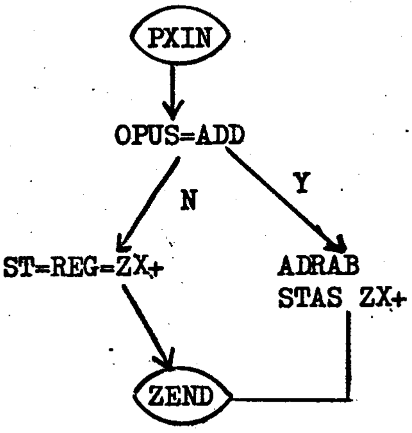

25
//SET THE STACK POINTER
ZREG & (=SC2=+=STRUCT=)
& = 0
Y
N
& < 16
Y
N
ADMX L&
(=SC1=) = 0
Y
N
PPO=ADD=A=I=&=0
PPPO=SET=A=I=&=0
ADRX A
ADRXA
3
Y
LABEL=0
N
PJMP==LABEL==F
ZEMD

PXIN=REG=OPUS

Load parameter values or addresses into the parameter space. 2 parameters.

1. REG: A or B or H or E

2. OPUS: ADD or not ADD .



PADR[1]=TYPE=OFFSET=LEVELS

Fetch absolute address into register A. 3 parameters.

1. TYPE:

C	for	constant
L	for	local
G	for	Global
V	for	variable parameter
N	for	non local
X	for	a non local variable

2. OFFSET: An integer n such that $-128 \le n < 128$

3. LEVELS: A non negative integer.

Code produced if TYPE is

```
L  or  C:  LDRA P;  ADDA L=OFFSET=

       G:  SETA P=LEVELS=;  ADDA L=OFFSET=

       V:  SETA P=OFFSET=

       N:  LDRYP;  LODP PØI } repeated =LEVELS= times
           LDRAP;  ADDA L=OFFSET=;  EXRYP

       X:  LDRYP;  LODP PØI } repeated =LEVELS= times
           SETA P=OFFSET=
           EXRYP
```

Note: See also PPPØ macro (section 5.1.1)

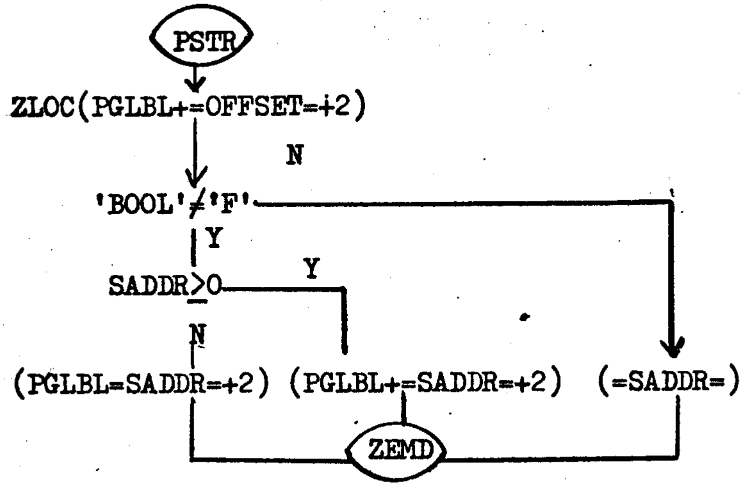
PSTR=OFFSET=SADDR=BOOL

Initialise globals. 3 parameters.

1. OFFSET: A positive unsigned integer.

2. SADDR: An integer; if non negative then unsigned.

3. BOOL: is T if SADDR represents an address and F it it is a constant value.



Notes:/

Notes: #PGLBL is the base of the stack frame.(see section 5.2). Two locations are reserved for the return address and the dynamic pointer of the main program.

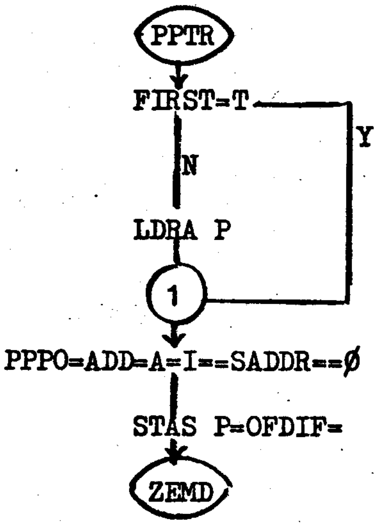
PPTR=OFDIF=SADDR=FIRST

Allocate space for arrays and records. 3 parameters.

1. OFDIF: An integer n such that $-128 \le n < 128$

2. SADDR: An integer.

3. FIRST: T if the macro is called for the first time.



Notes: =FIRST= was introduced to optimise the code when this macro is called successively.

PALC=TYPE=OFFSET=LEVELS=LENGTH

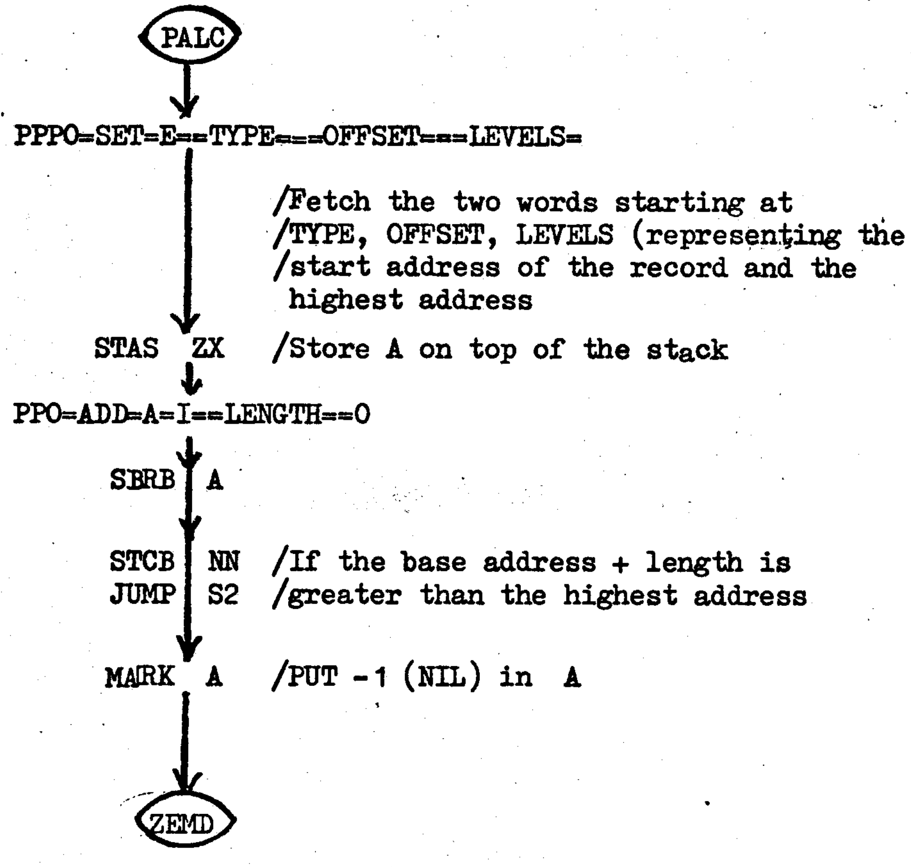
Dynamic allocation of records. 4 parameters.

1. TYPE: Only C, T, L, G, N, X, Z as in PPPO (section 5.1.1)

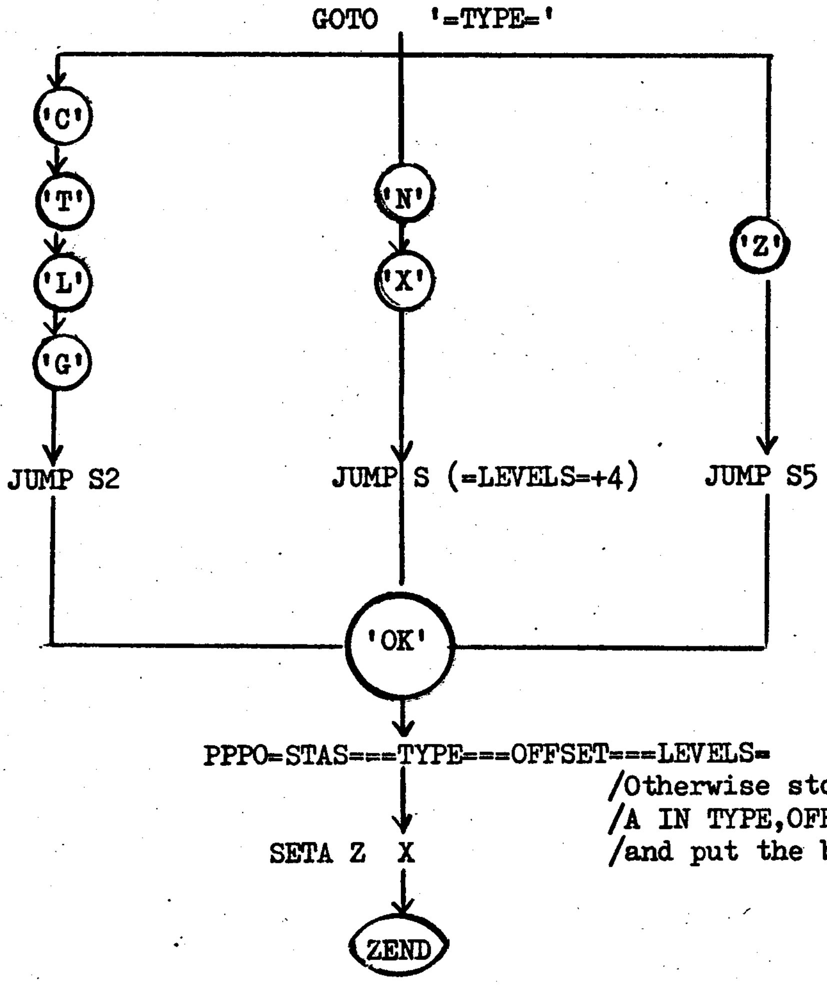
2. OFFSET: As in PPPØ (see section 5.1.1)

3. LEVELS: As in PPPØ (see section 5.1.1)

4. LENGTH: A positive integer, the length of the record.



```
                 PALC
PPPO=SET=E==TYPE===OFFSET===LEVELS=
                      /Fetch the two words starting at
                      /TYPE, OFFSET, LEVELS (representing the
                      /start address of the record and the
                       highest address
      STAS  ZX        /Store A on top of the stack
PPO=ADD=A=I==LENGTH==0
      SBRB  A
      STCB  NN        /If the base address + length is
      JUMP  S2        /greater than the highest address
      MARK  A         /PUT -1 (NIL) in A
                 ZEMD
```



/Otherwise store the contents of
/A IN TYPE,OFFSET, LEVELS
/and put the base address in A

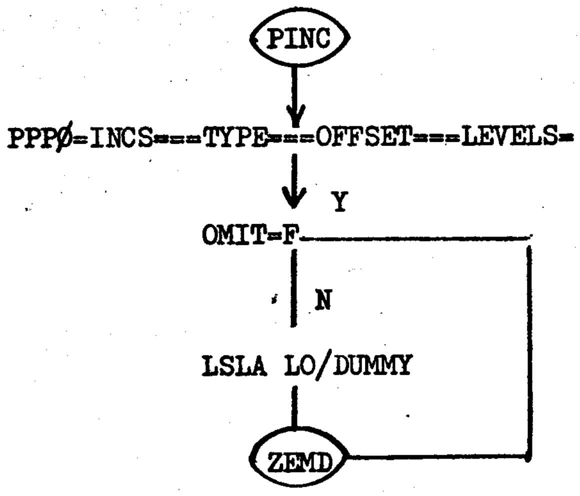
5.1.6 Macros for Repetitive Commands

These include increment a datum by one/skip if zero PINC, decrement by one/skip if zero and code for the FOR statement PFOR.

PINC=TYPE=OFFSET=LEVELS=OMIT

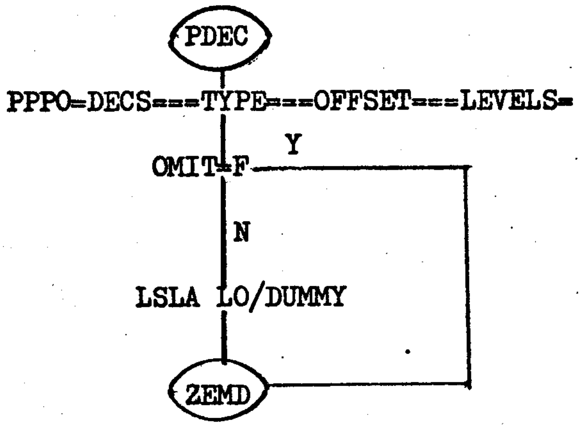
Increment a datum. 4 parameters.

1. TYPE: As in PPPØ (see section 5.1.1)
2. OFFSET: As in PPPØ (see section 5.1.1)
3. LEVELS: As in PPPØ (see section 5.1.1)
4. OMIT: if F then no dummy instruction follows the < increment and skip > command.



PDEC=TYPE=OFFSET=LEVELS=OMIT

Decrement a datum. Same comments as PINC (section 5.1.6)



Notes: PINC and PDEC produce optimised code for the <FOR> statement when its bounds are constants. This statement in its full generality is dealt with in the following macro (PFOR):

PFOR[1]=TYPE=OFFSET=LEVELS=MHR=LABEL=ALTER1=ALTER2=OMIT

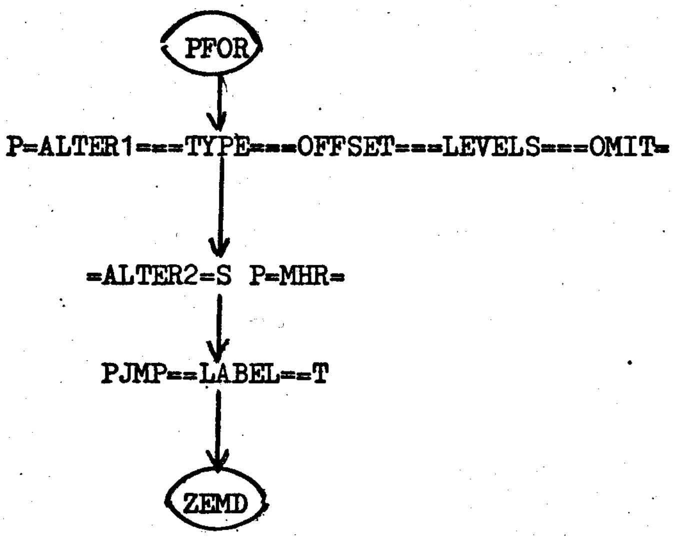
1-3: As for PPPØ

4: MHR: An integer n : $-128 \le n < 128$

5: LABEL: A positive unsigned integer

6,7. ALTER 1
ALTER 2: INC , DEC or DEC , INC

8. OMIT: As for the PINC and PDEC macros (5.1.6)

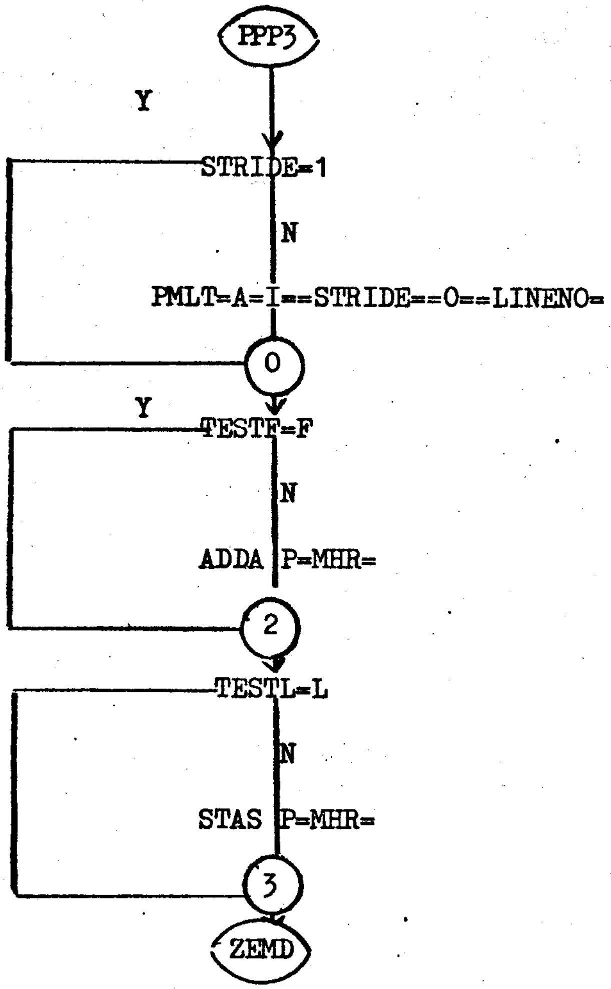


5.1.7 ARRAY MANIPULATION MACROS

The macro for accessing subscripted variables are PIND and POND. PPP3 contains theircommon logic.For an n dimensional array $(n>1)$, n calls of PIND or POND are performed. PCHR is called to compare two elements of character arrays.

PPP3=STRIDE=TESTF=MHR=TESTL=LINENO

1. STRIDE: $\{S \in I \mid S \geq 1\}$
2. TESTF: F for first, (S for subsequent)
3. MHR: $\{n \in I \mid 5 \leq n < 128\}$
4. TESTL: L for lost (P if not)
5. LINENO: If non zero, then check for overflow in fix point multiplication is performed.



Note: For the exact meaning of TESTF and TESTL refer to the corresponding chapter by R. Cupples.

PIND=TYPE=OFFSET=LEVELS=STRIDE=TESTF=MHR=TESTL=LO=HI=LINENO

For accessing elements of Multi-dimensional arrays. 10 parameters.

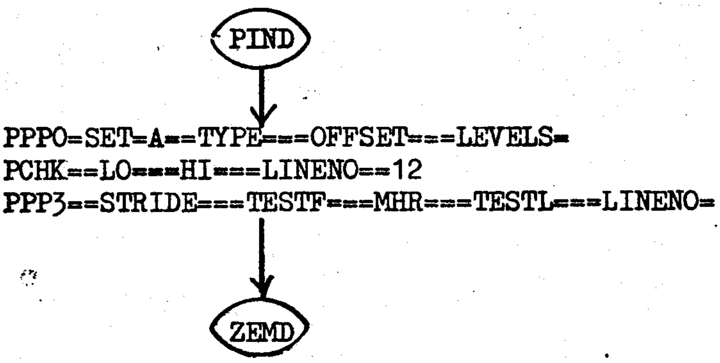
1,2,3: As for PPPO (section 5.1.1)

4,5,6,7: As for PPP3 (section 5.1.7)

8. LO: The lower bound

9. HI: The upper bound

10. LINENO: If non zero, then the line number of the corresponding Pascal statement.



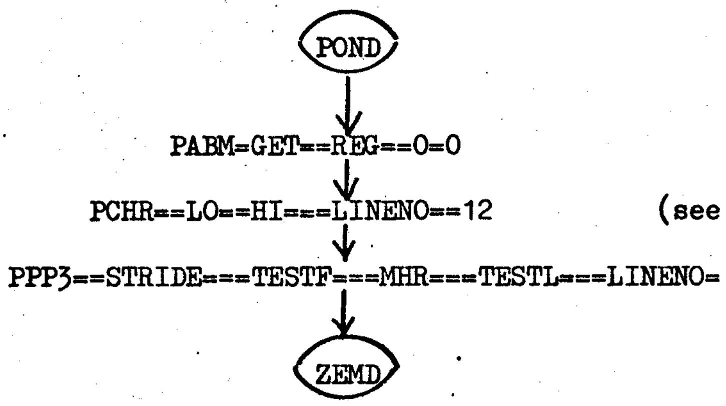
```
PPPO=SET=A==TYPE===OFFSET===LEVELS=
PCHK==LO===HI===LINENO==12
PPP3==STRIDE===TESTF===MHR===TESTL===LINENO=
```

POND=REG=STRIDE=TESTF=MHR=TESTL=LO=HI=LINENO

For accessing array elements when one or more subscripts are also subscripted variables. 8 parameters.

1. REG: A register name (A or H)

2-8: As PIND (section 5.1.7)



```
PABM=GET==REG==0=0
PCHR==LO==HI===LINENO==12          (see 5.1.9)
PPP3==STRIDE===TESTF===MHR===TESTL===LINENO=
```

2
PCHR=MHR

For comparing elements of two character arrays. 1 parameter.

1. MHR: $\{n \in I \mid 5 \leq n < 128\}$

A single element is generated:

```
SUBA P=MHR=
```

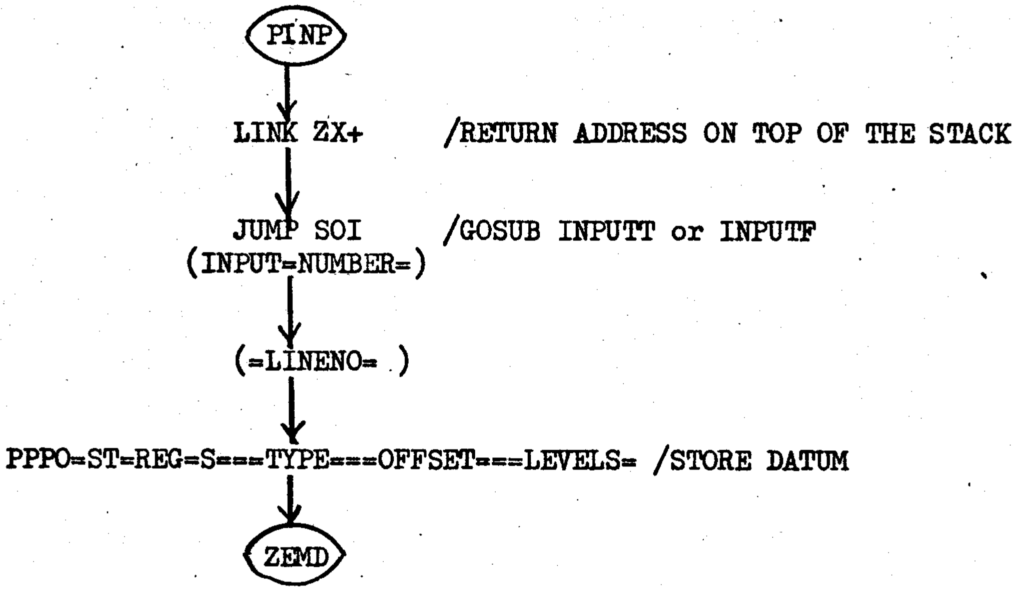
5.1.8 I/O Macros

These include input a character or a number PINP, output a character or a number POUT, and output a character string PTXT.

PINP[1]=REG=TYPE=OFFSET=LEVELS=NUMBER=LINENO

Input a character or a number. 6 parameters.

1. REG:	A or H only
2,3,4:	As in PPPO (5.1.1)
5. NUMBER:	T for inputing a number, F for a character.
6. LINENO:	The line number of the corresponding READ command in the Pascal source.



POUT=REG=TYPE=OFFSET=LEVELS=NUMBER=CHAN=MHR=LINENO

Output a character or a number. 8 parameters.

1. REG: A or H only.

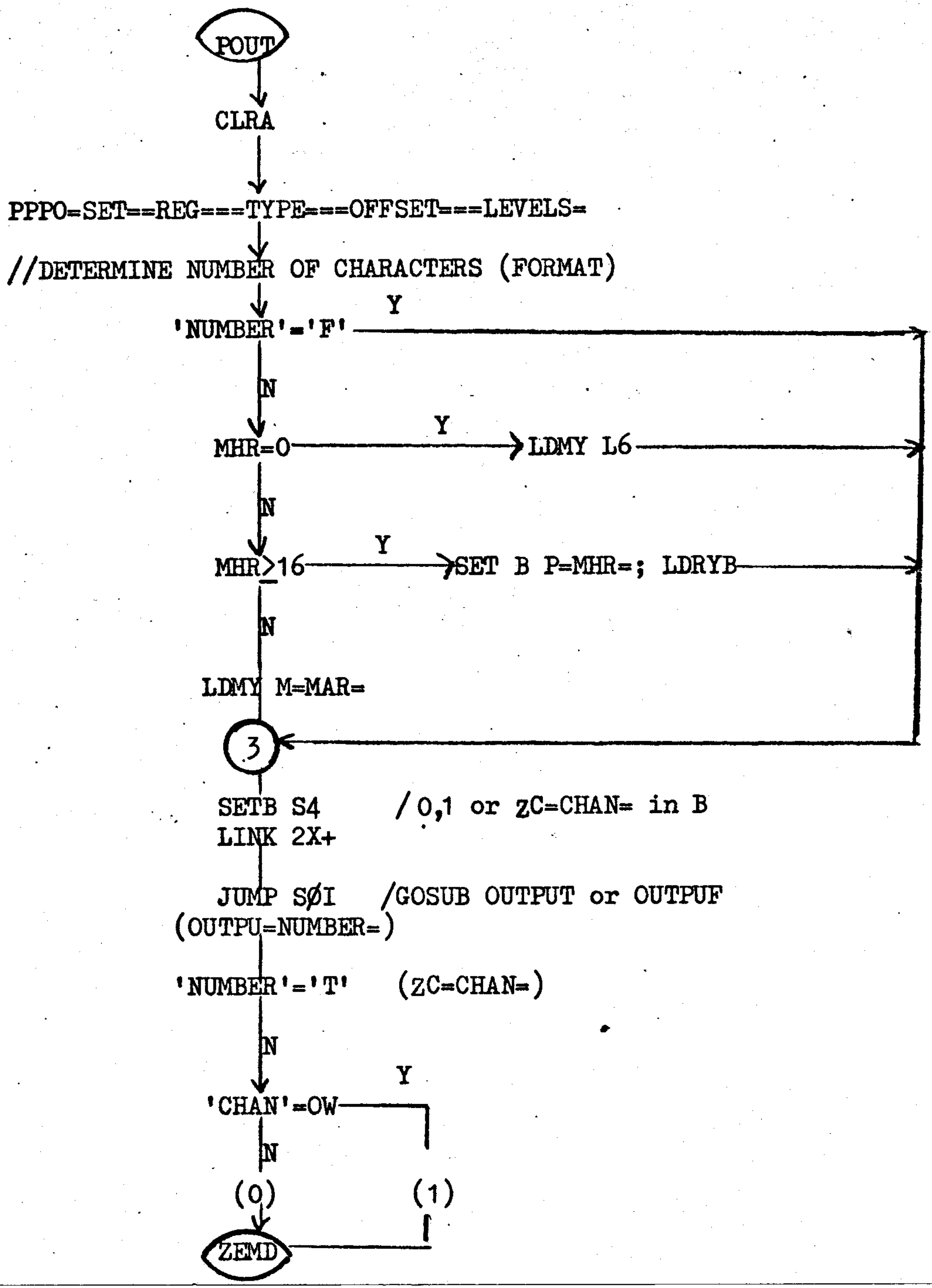
2,3,4: As in PPPØ, section 5.1.1

5. NUMBER: T for number and F for character output

6. CHAN: OU for channel SC2 (line printer) and OW for channel SC3 (disc). See also section 5.2.3.

7. MHR: if Ø then the number to be printed is output with the default number of places (6). Otherwise it represents P relative location, containing the number of significant places.

8. LINENO: The line number of the corresponding WRITE command in the Pascal source.



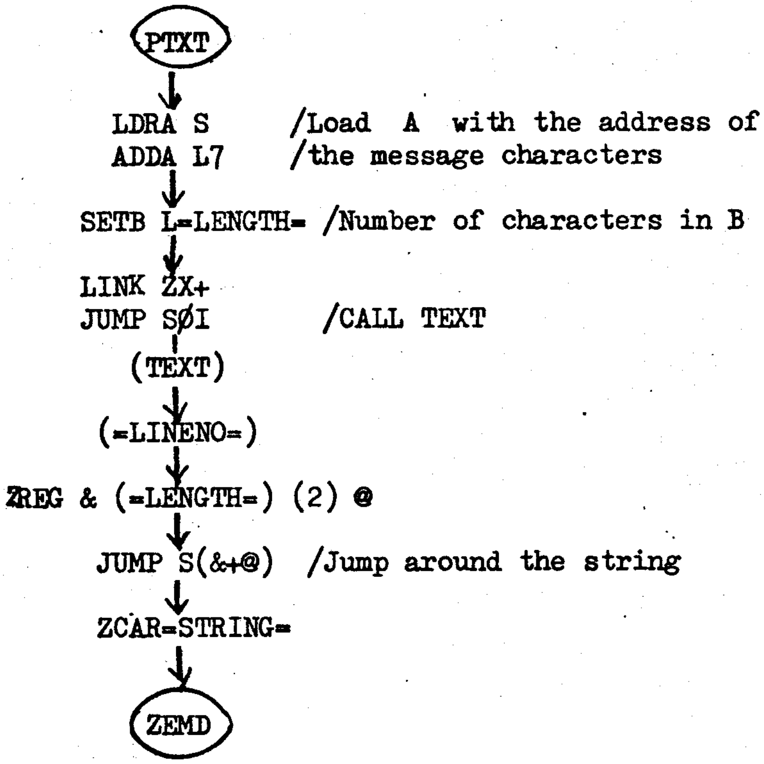
PTXT[1]=LENGTH=STRING=LINENO .

Output a character string. 3 parameters. The assembler registers & and @ are used.

1. LENGTH: The number n of characters in the string: 1≤n≤250

2. STRING: The character string. This should not contain = or ; or CR or LF .

3. LINENO: The line number of the corresponding TEXT command in the Pascal source.



```
              PTXT
                ↓
          LDRA S      /Load A with the address of
          ADDA L7     /the message characters
                ↓
        SETB L=LENGTH= /Number of characters in B
                ↓
        LINK ZX+
        JUMP SØI        /CALL TEXT
                ↓
            (TEXT)
                ↓
          (=LINENO=)
                ↓
ZREG & (=LENGTH=) (2) @
                ↓
          JUMP S(&+@)  /Jump around the string
                ↓
          ZCAR=STRING=
                ↓
              ZEMD
```

5.1.9 Run-time errors

PCHK[1]=LO=HI=LINENO=MESS

Run-time check. 4 parameters.

1. LO: The lowest allowable value.

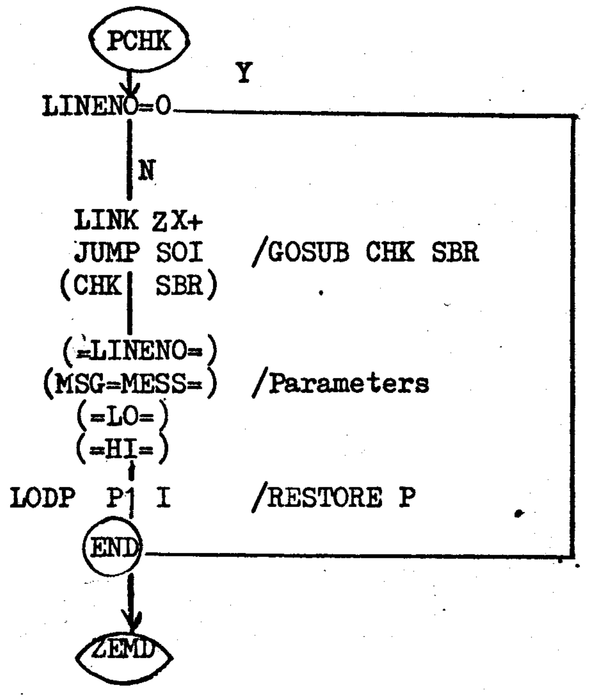
2. HI: The highest allowable value.

3. LINENO: The line number of the corresponding Pascal statement

4. MESS: The message number:

Message numbers:	Message:
2	Overlength number in input transfer.
4	Case index outside bounds.
5	Overflow in fix point multiplication.
6	Invalid character 'the character' in input transfer.
7	Input transfer failure (device unavailable or no data).
8	Output transfer failure (device unavailble).
9	Case condition undefined.
10	Violation.
11	Value assigned to subrange variable out of range.
12	Array index out of bounds.
13	Integer division by zero.
14	Run-time stack exhausted.

Messages 0, 1 and 3 are used by the run-time monitor (see section 5.2).

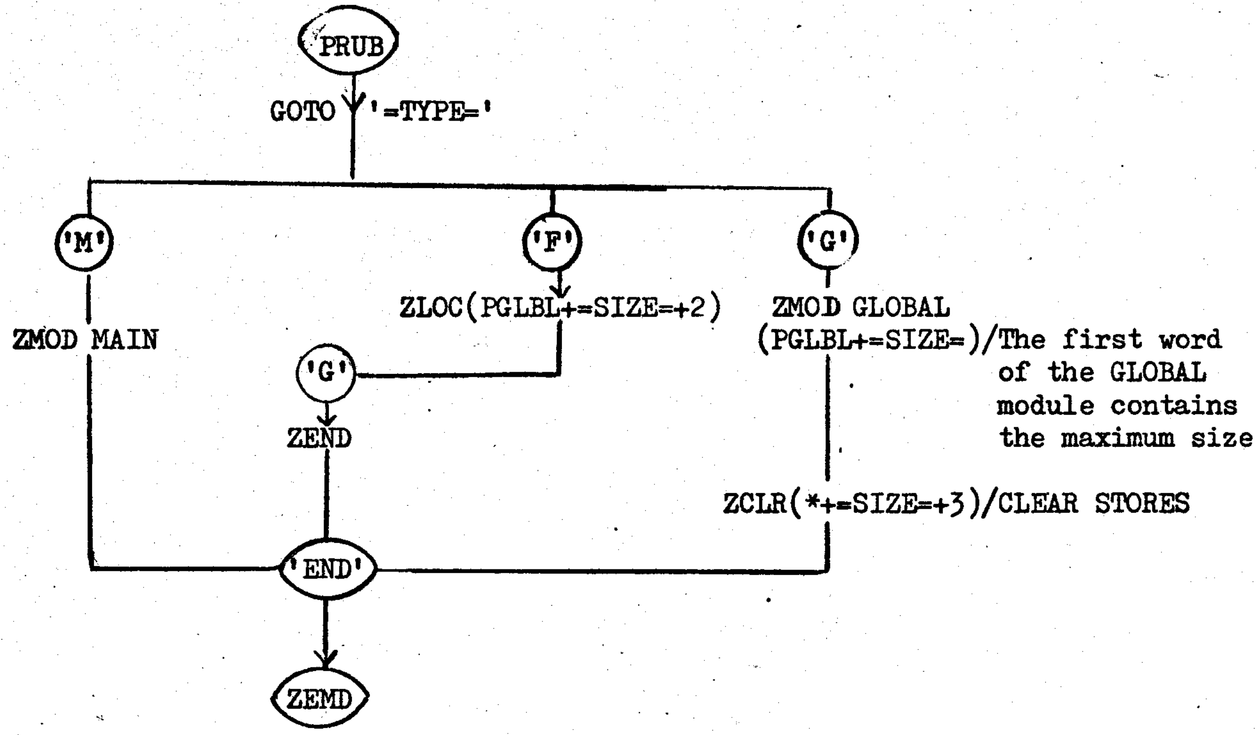


PRUB^2=TYPE=SIZE

This enables the compiler to specify the maximum size the object program is to occupy at run-time. 2 parameters.

1. TYPE: M for the start of the MAIN module.
E at the end of the MAIN module.
G for the start of the GLOBAL module.
F at the end of the GLOBAL module.

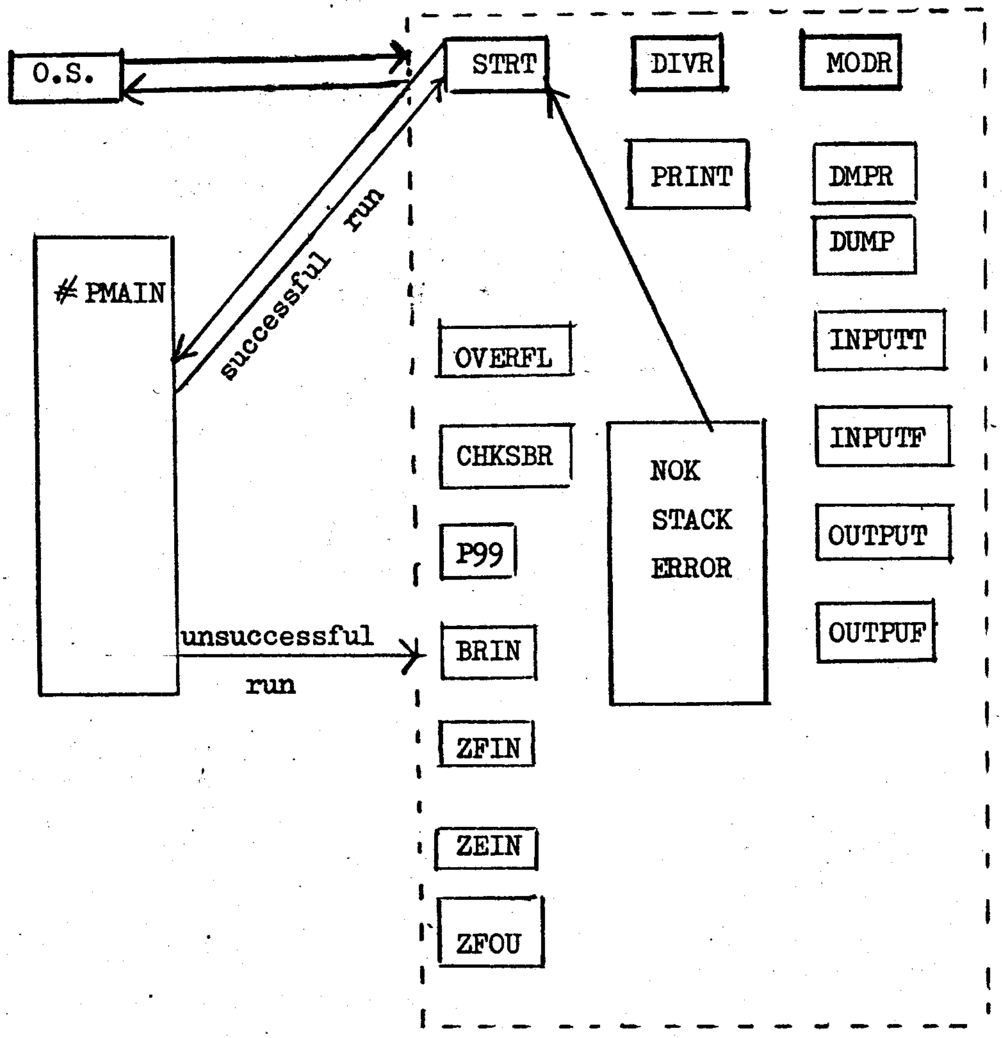
2. SIZE: The size of the GLOBAL module in words


PRUB
GOTO
'=TYPE='
'M'
'F'
'G'
ZMOD MAIN
ZLOC(PGLBL+=SIZE=+2)
ZMOD GLOBAL
(PGLBL+=SIZE=)/The first word of the GLOBAL module contains the maximum size
'G'
ZEND
ZCLR(*+=SIZE=+3)/CLEAR STORES
'END'
ZEMD

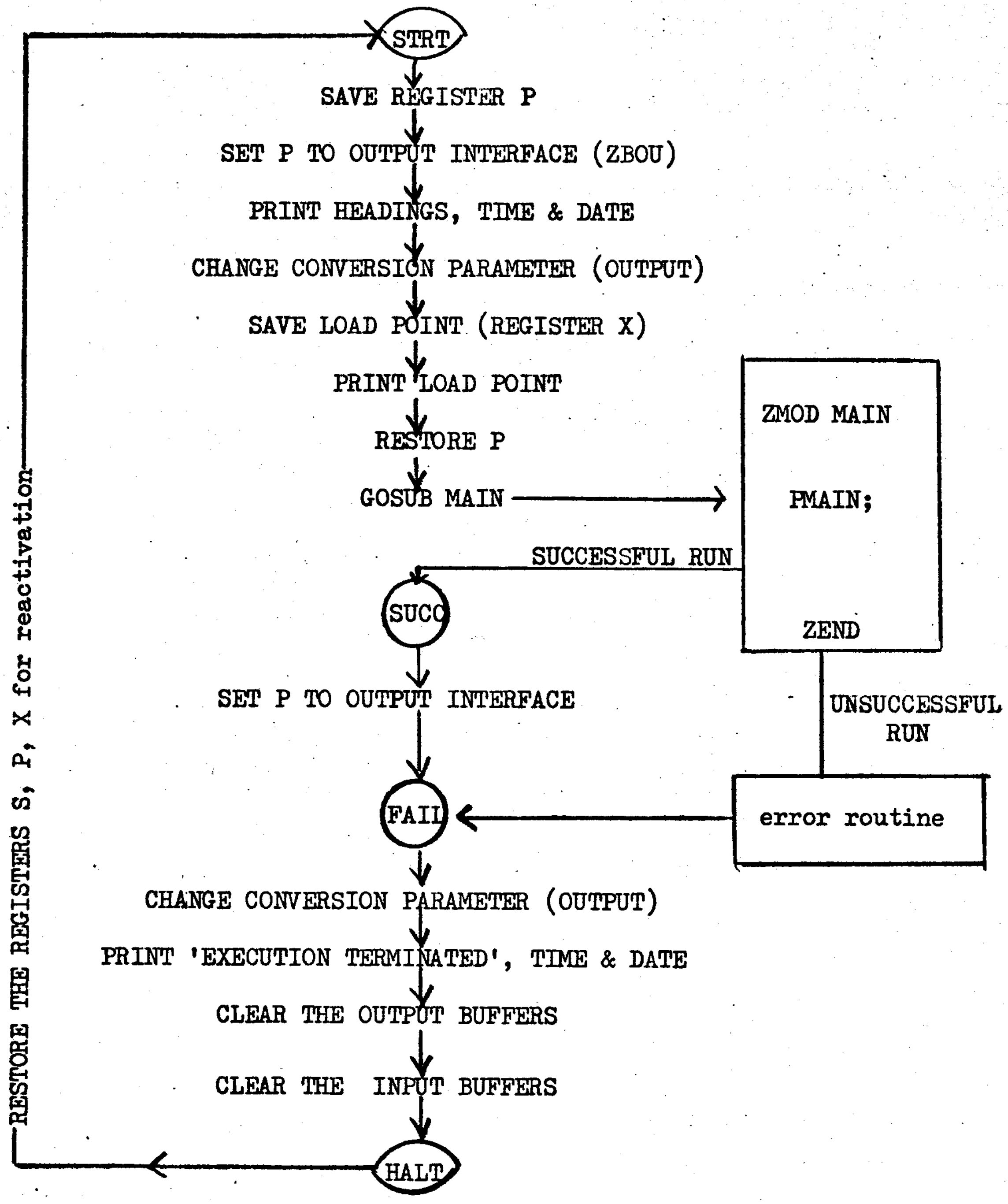
5.2 RUN TIME MONITOR

The purpose of the run time monitor is to provide a standard interface between the Pascal compiler and the operating system.

The run time monitor activates the Pascal program and control returns to it after execution. It contains the common I/O subroutines, i.e. PRINT, INPUTT, INPUTF, OUTPUT, OUTPUF, the run time error recovery routines i.e. OVERFL, CHRSBR, ERROR, BRIN, P99, ZFIN, ZEIN, ZFOU and the utility subroutines. DMPR and DUMP. Section 5.2.1 describes its structure, 5.2.2 its subroutines and 5.2.3 the integration layout.

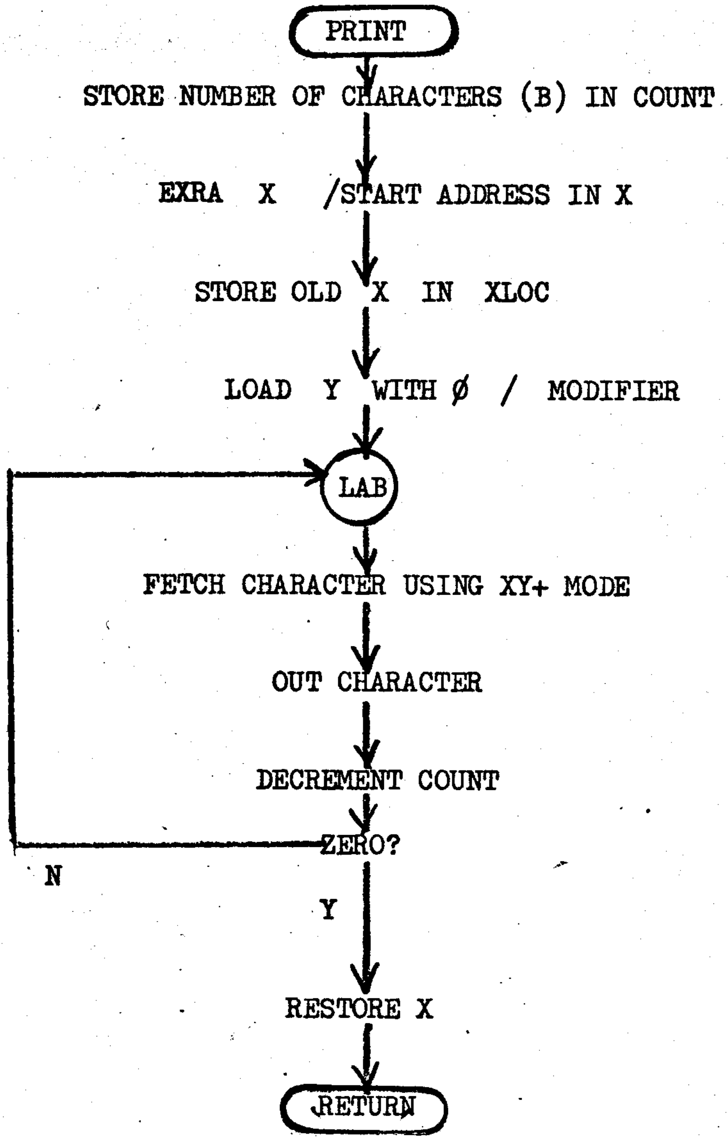


5.2.1 MAIN PROGRAM (STRT)


STRT
SAVE REGISTER P
SET P TO OUTPUT INTERFACE (ZBOU)
PRINT HEADINGS, TIME & DATE
CHANGE CONVERSION PARAMETER (OUTPUT)
SAVE LOAD POINT (REGISTER X)
PRINT LOAD POINT
RESTORE P
GOSUB MAIN
ZMOD MAIN
PMAIN;
ZEND
SUCCESSFUL RUN
SUCC
SET P TO OUTPUT INTERFACE
UNSUCCESSFUL RUN
error routine
FAIL
CHANGE CONVERSION PARAMETER (OUTPUT)
PRINT 'EXECUTION TERMINATED', TIME & DATE
CLEAR THE OUTPUT BUFFERS
CLEAR THE INPUT BUFFERS
HALT
RESTORE THE REGISTERS S, P, X for reactivation

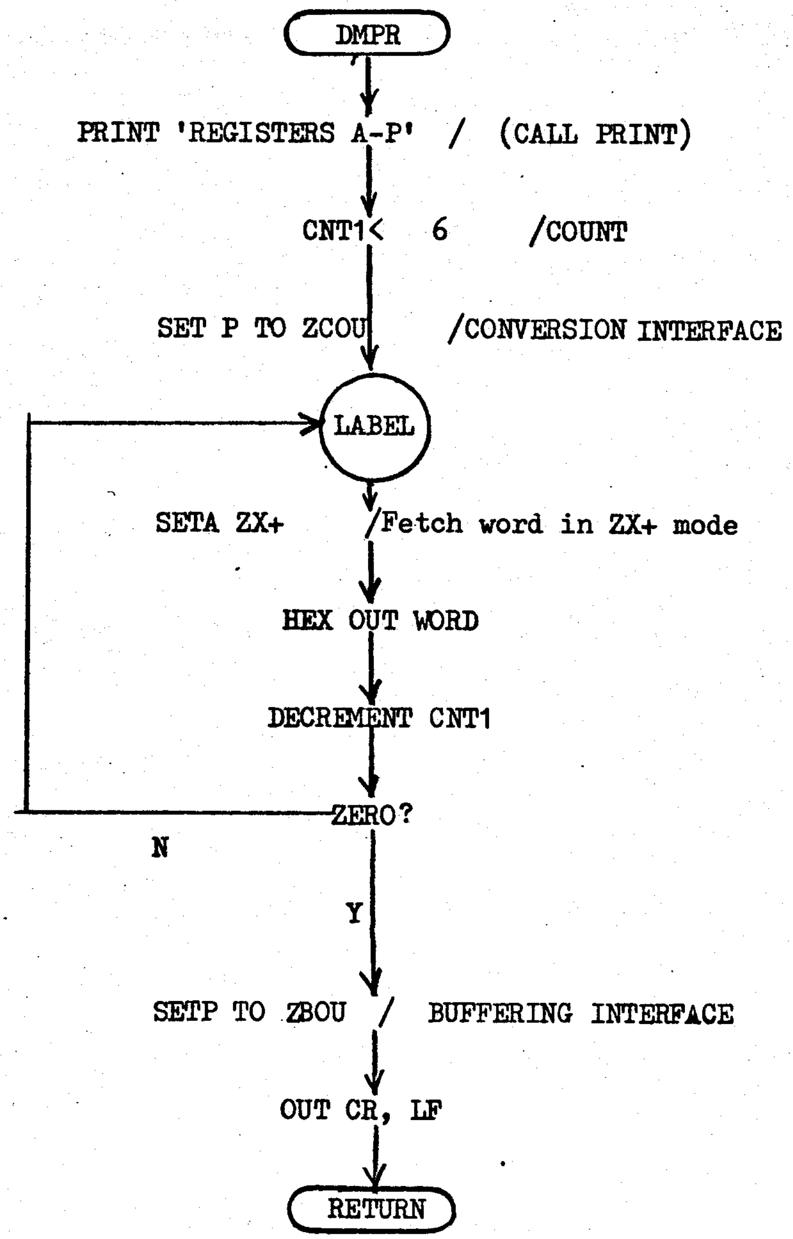
5.2.2 SUBROUTINES

#PRINT: On entry A contains the address of the message characters, B the number of characters, P points to ZBOU. On exit A, B & Y are destroyed.



#DMPR:

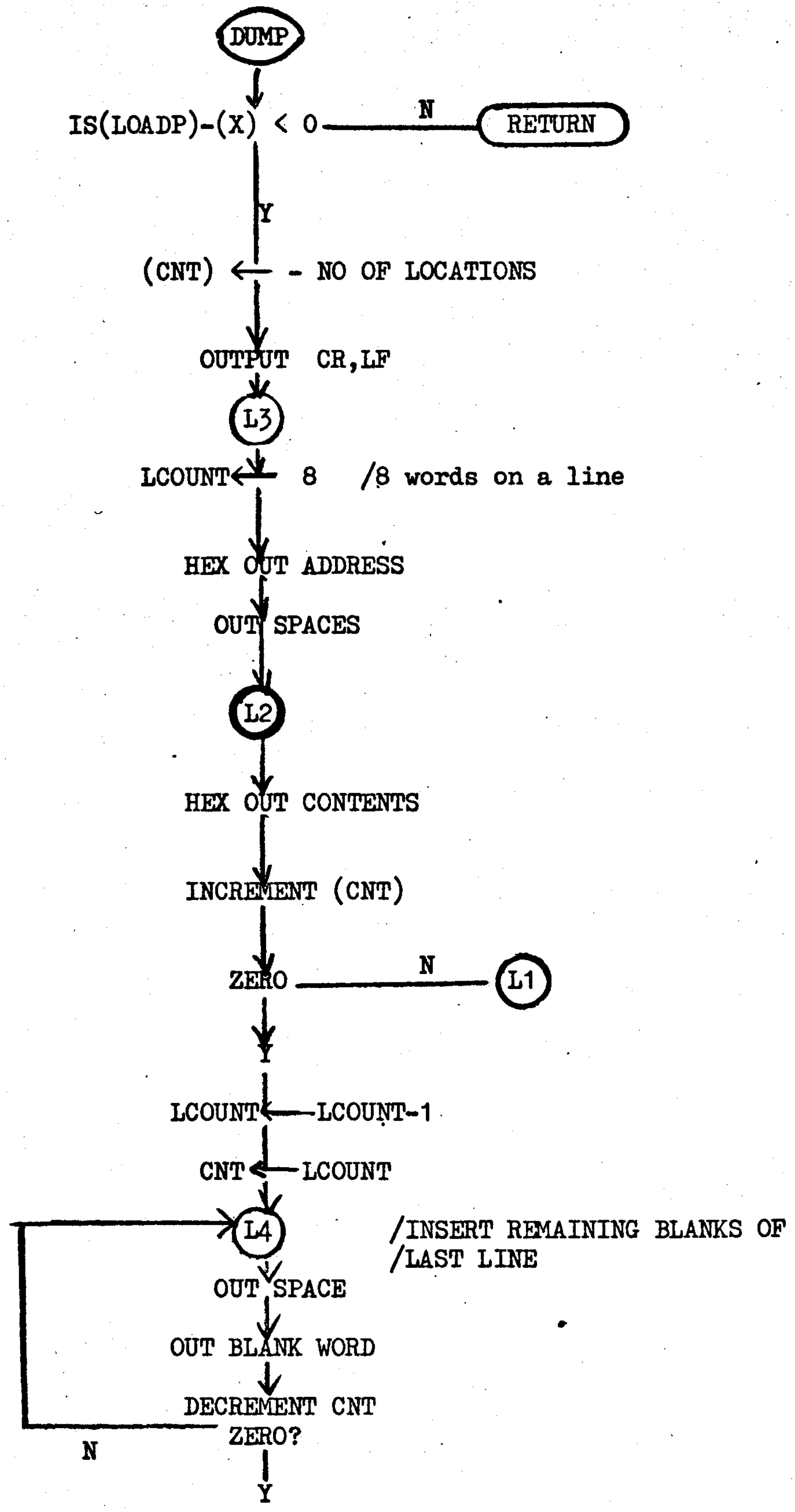
Dump the top six locations of the stack. These should contain the registers A, B, X, Y, S and P. X should point to the location containing register A. P should point to ZBOU. A & B & Y are destroyed on exit.



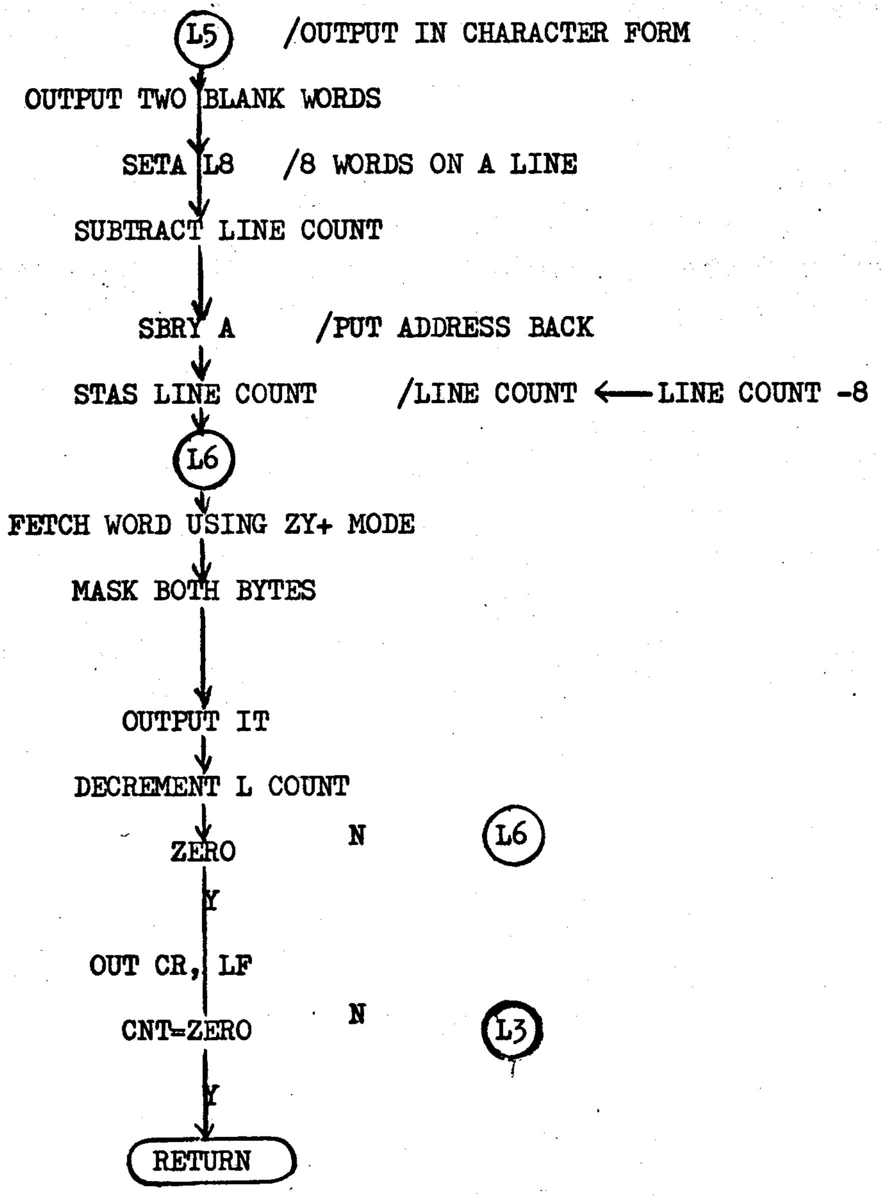
DMPR
PRINT 'REGISTERS A-P' / (CALL PRINT)
CNT1< 6 /COUNT
SET P TO ZCOU /CONVERSION INTERFACE
LABEL
SETA ZX+ /Fetch word in ZX+ mode
HEX OUT WORD
DECREMENT CNT1
ZERO?
N
Y
SETP TO ZBOU / BUFFERING INTERFACE
OUT CR, LF
RETURN

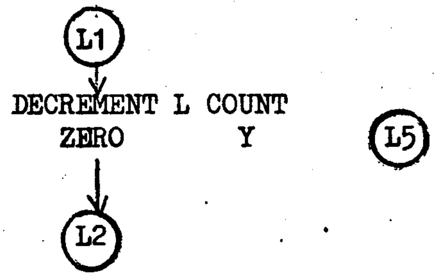
#DUMP:

This subroutine dumps the memory locations from the address given in LOADP to the current contents of register X . On entry P should point to ZBOU (buffering interface). On exit A & B & Y are destroyed



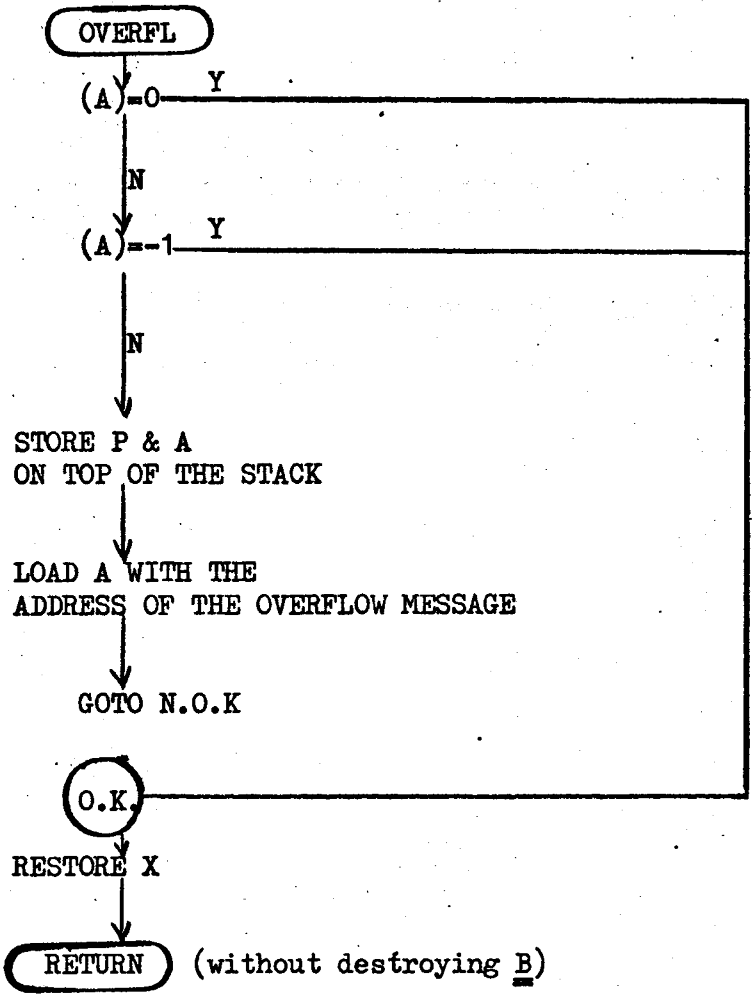
/


L5
/OUTPUT IN CHARACTER FORM
OUTPUT TWO BLANK WORDS
SETA L8
/8 WORDS ON A LINE
SUBTRACT LINE COUNT
SBRY A
/PUT ADDRESS BACK
STAS LINE COUNT
/LINE COUNT ← LINE COUNT -8
L6
FETCH WORD USING ZY+ MODE
MASK BOTH BYTES
OUTPUT IT
DECREMENT L COUNT
ZERO
N
L6
Y
OUT CR, LF
CNT=ZERO
N
L3
Y
RETURN


L1
DECREMENT L COUNT
ZERO
Y
L5
L2

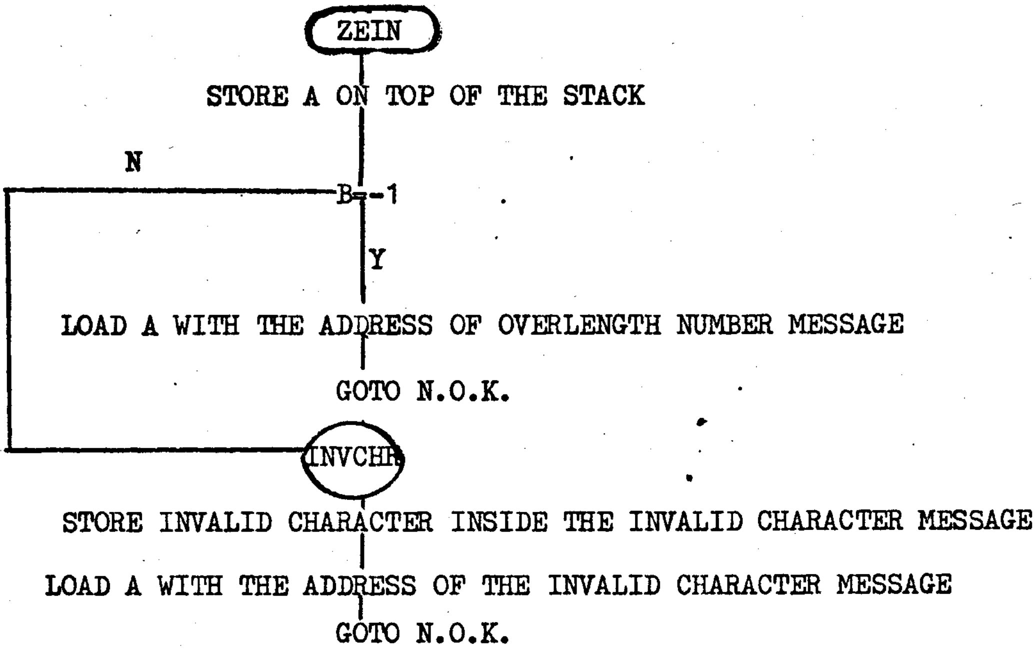
#OVERFL;

Overflow routine. This is called by the PD2S macro (section 5.1.3)



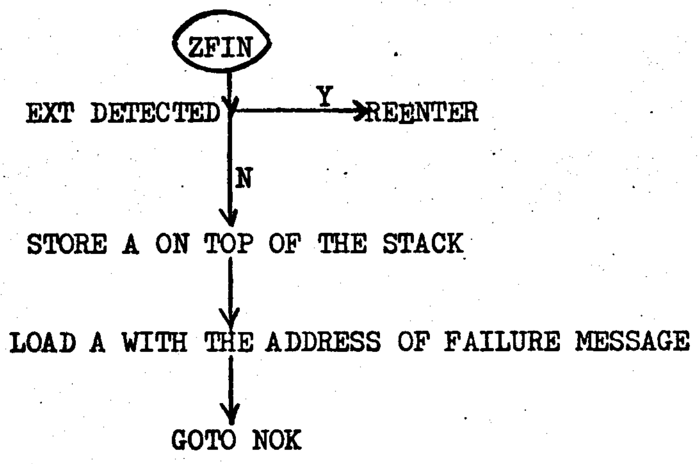
#ZEIN;

Control passes to this subroutine if an invalid character or an overlength number is encountered on input. See [2].


ZEIN
STORE A ON TOP OF THE STACK
N
B=-1
Y
LOAD A WITH THE ADDRESS OF OVERLENGTH NUMBER MESSAGE
GOTO N.O.K.
INVCHR
STORE INVALID CHARACTER INSIDE THE INVALID CHARACTER MESSAGE
LOAD A WITH THE ADDRESS OF THE INVALID CHARACTER MESSAGE
GOTO N.O.K.

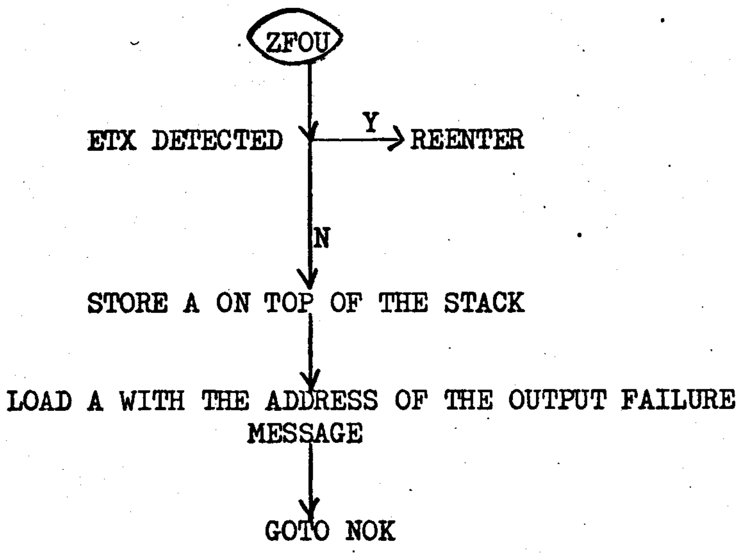
#ZFIN:

Control passes to this subroutine if any of the status bits are set on input [2]


ZFIN
EXT DETECTED
Y
REENTER
N
STORE A ON TOP OF THE STACK
LOAD A WITH THE ADDRESS OF FAILURE MESSAGE
GOTO NOK

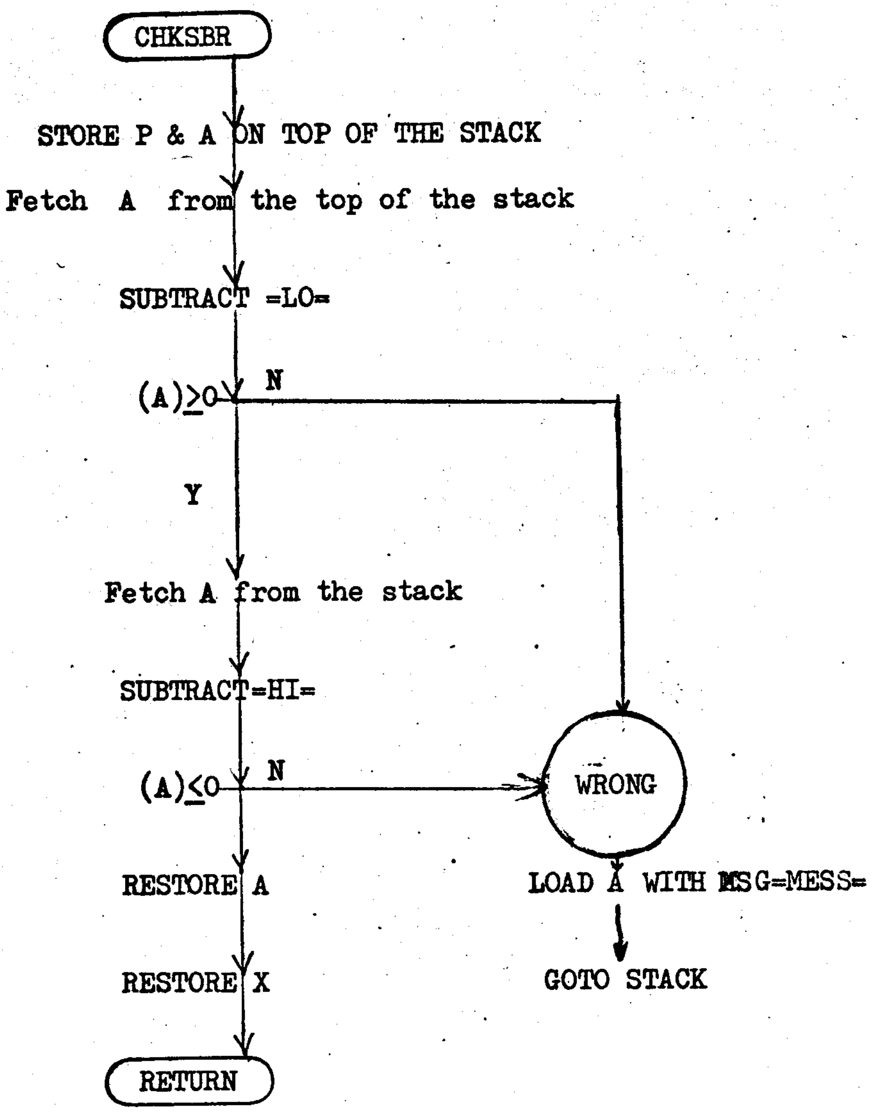
#ZFOU;

Control passes to this subroutine if any of the status bits are set on output [2]


ZFOU
ETX DETECTED
Y
REENTER
N
STORE A ON TOP OF THE STACK
LOAD A WITH THE ADDRESS OF THE OUTPUT FAILURE MESSAGE
GOTO NOK

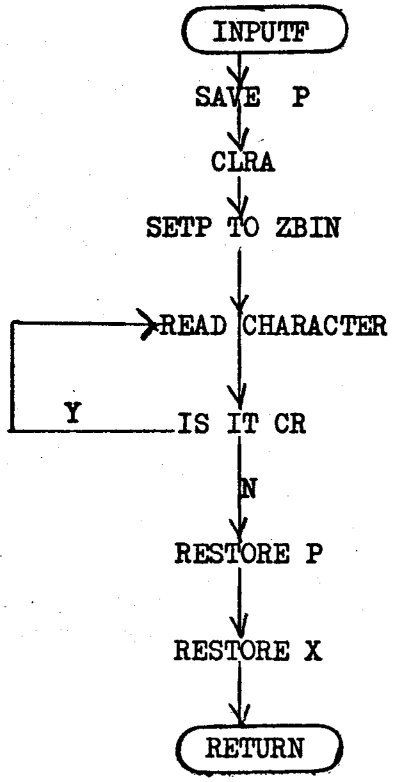
#CHKSBR;

This subroutine is entered when the PCHK macro (section 5.1.9) is called with LINENO≠Ø .



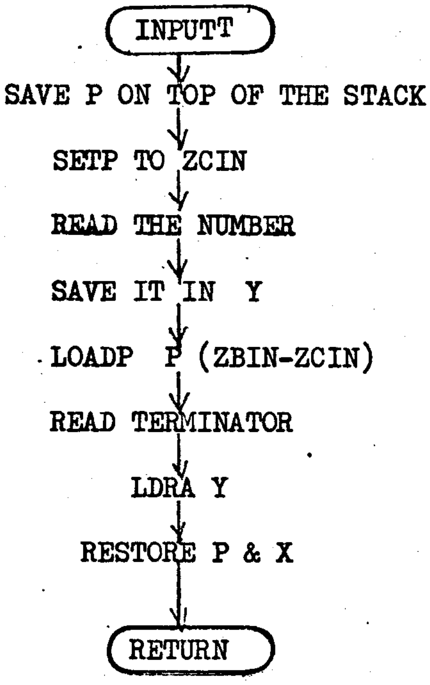
#INPUTF;

Input a character. It is called by the PINP macro with NUMBER=F (section 5.1.8).



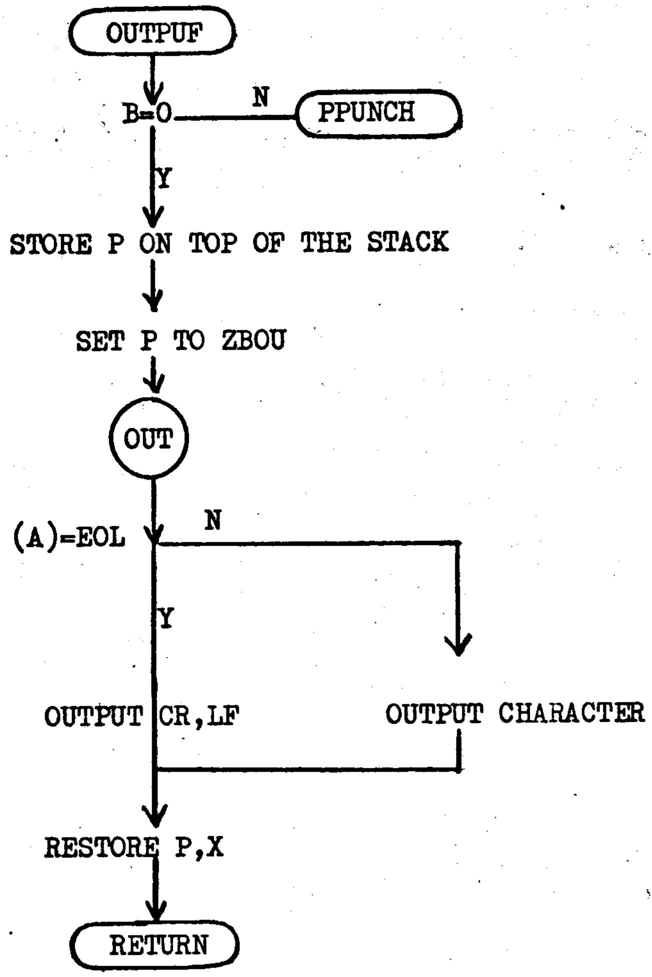
#INPUTT;

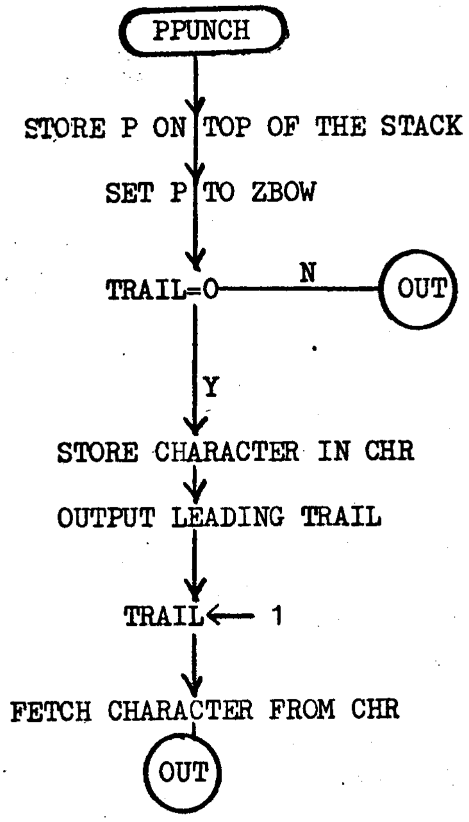
Read a number. It is called by the PINP macro when NUMBER=T (section 5.1.8 [2])



#OUTPUF;

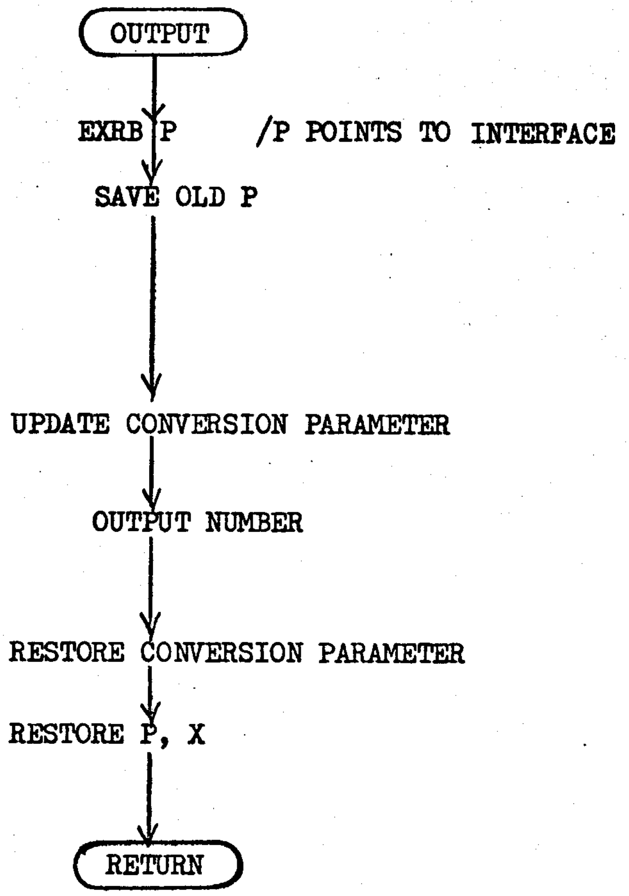
Output a character. This is called from the POUT macro call with NUMBER=F. On entry B contains Ø for channel SC2 or 1 for channel SC3.





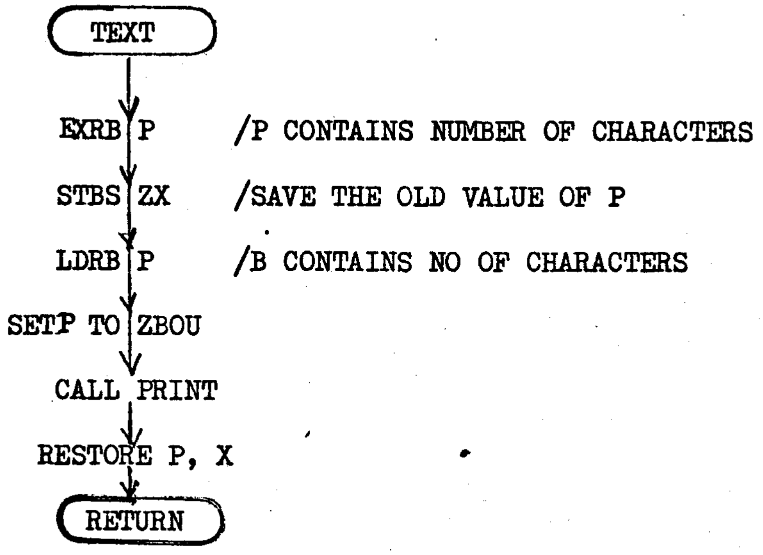
#OUTPUT;

Output a number. See POUT macro (section 5.1.8) with NUMBER=T. On entry Y contains number of characters (field width).



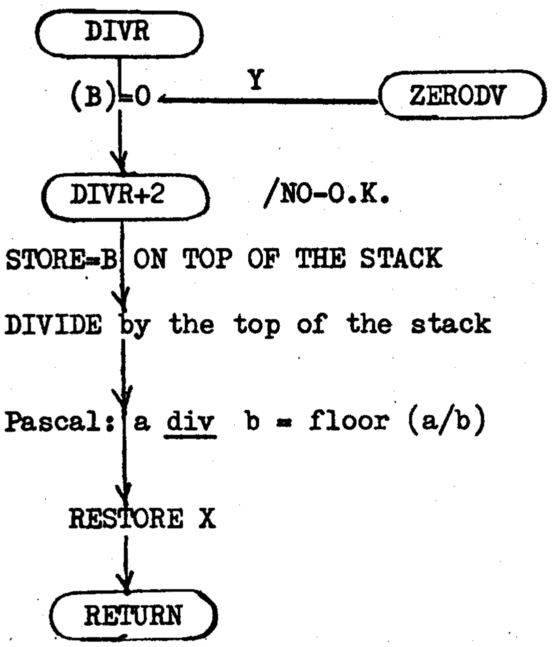
#TEXT;

See TEXT macro (section 5.1.8)



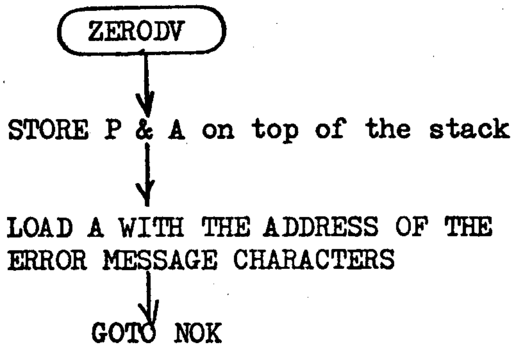
#DIVR;

See PPP5 macro, section 5.1.3


DIVR
(B)=0
Y
ZERODV
DIVR+2
/NO-O.K.
STORE=B ON TOP OF THE STACK
DIVIDE by the top of the stack
Pascal: a div b = floor (a/b)
RESTORE X
RETURN

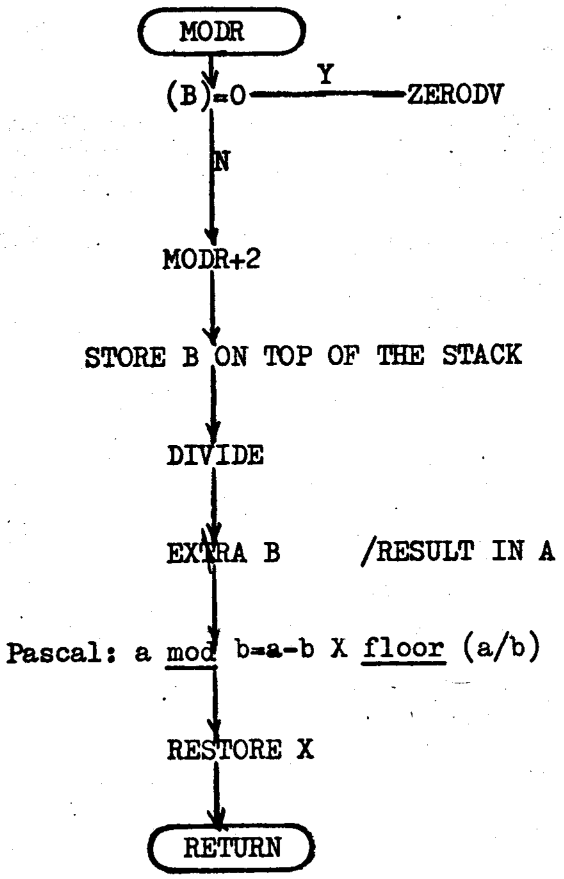
#ZERODV;

Division by zero


ZERODV
STORE P & A on top of the stack
LOAD A WITH THE ADDRESS OF THE
ERROR MESSAGE CHARACTERS
GOTO NOK

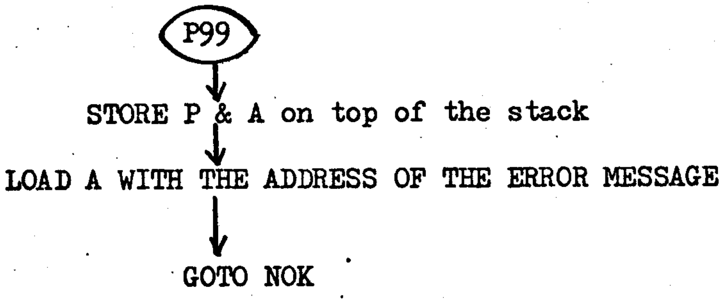
#MODR;

See PPP5 (section 5.1.3)



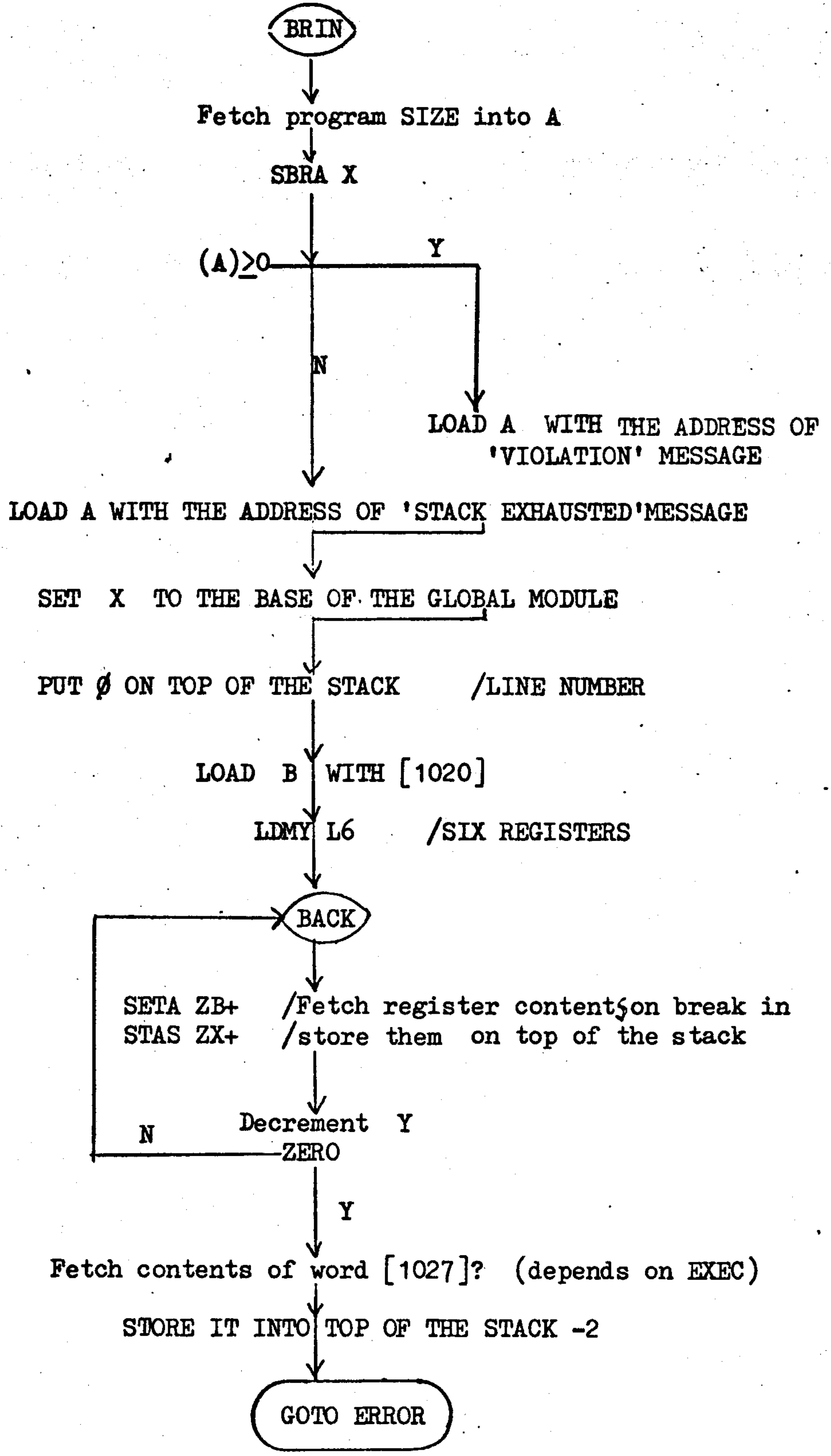
#P99:

Control passes to this routine if an undefined case condition arises (see PCAS section 5.1.4)


P99
STORE P & A on top of the stack
LOAD A WITH THE ADDRESS OF THE ERROR MESSAGE
GOTO NOK

#BRIN;

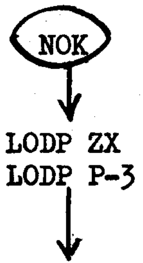
Break in routine


BRIN
Fetch program SIZE into A
SBRA X
(A)≥0
Y
N
LOAD A WITH THE ADDRESS OF 'VIOLATION' MESSAGE
LOAD A WITH THE ADDRESS OF 'STACK EXHAUSTED'MESSAGE
SET X TO THE BASE OF THE GLOBAL MODULE
PUT Ø ON TOP OF THE STACK /LINE NUMBER
LOAD B WITH [1020]
LDMY L6 /SIX REGISTERS
BACK
SETA ZB+ /Fetch register contents on break in
STAS ZX+ /store them on top of the stack
Decrement Y
ZERO
N
Y
Fetch contents of word [1027]? (depends on EXEC)
STORE IT INTO TOP OF THE STACK -2
GOTO ERROR

NOK; STACK; ERROR;

This is the general failure routine. It has three entry points.

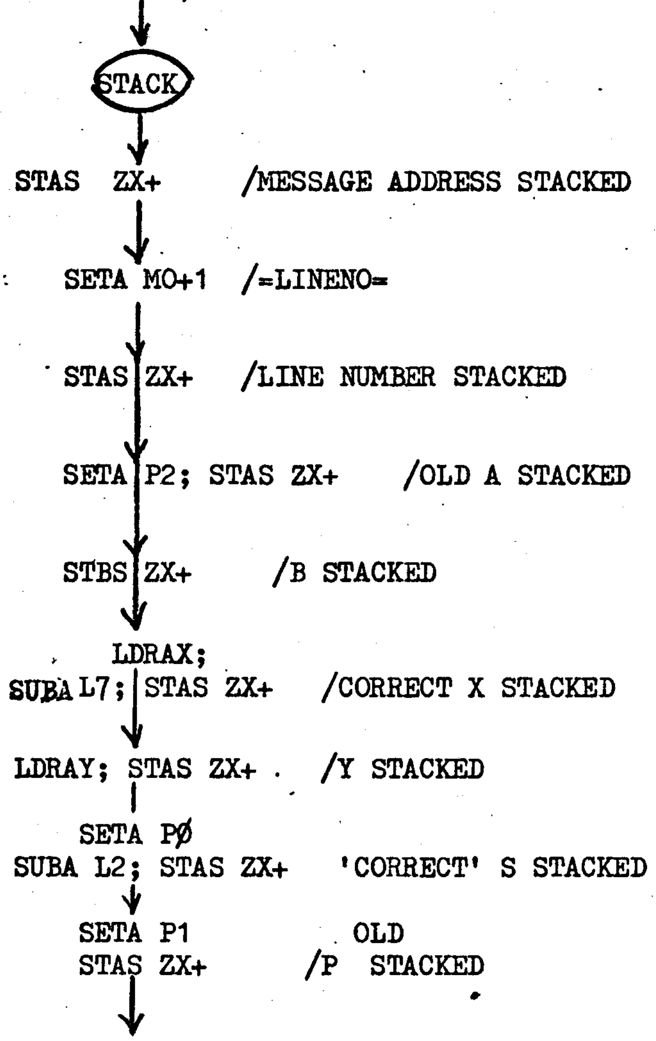
On entry to NOK the stack should contain S,P and the old contents of register A. Register A should contain the address of a message



```
LODP ZX
LODP P-3
```

This NOK sets P to the top of the stack -3 .

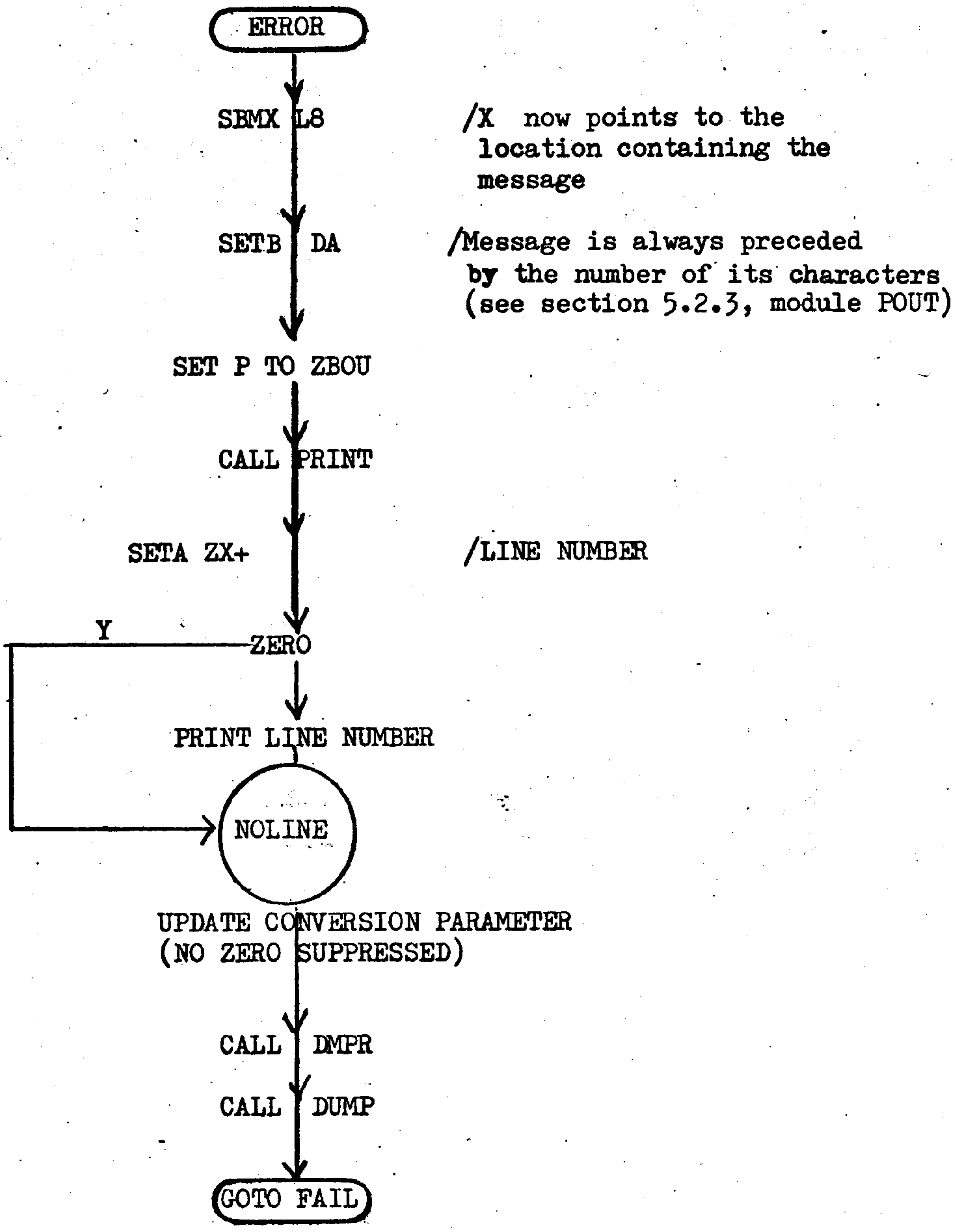
On entry to STACK, the stack should contain S,P and the old contents of register A. Register A contains the address of a message. Register P should point to the location containing S(top of the stack minus three).



```
STAS   ZX+       /MESSAGE ADDRESS STACKED

   SETA MO+1     /=LINENO=

   STAS ZX+      /LINE NUMBER STACKED

   SETA P2; STAS ZX+    /OLD A STACKED

   STBS ZX+        /B STACKED

     LDRAX;
SUBA L7; STAS ZX+    /CORRECT X STACKED

LDRAY; STAS ZX+ .   /Y STACKED

    SETA PØ
SUBA L2; STAS ZX+   'CORRECT' S STACKED

    SETA P1           OLD
    STAS ZX+        /P  STACKED
```

On/

On entry to ERROR, the stack should contain the address of a message, a line number and the registers in the order A to P .


ERROR
SBMX L8
/X now points to the location containing the message
SETB DA
/Message is always preceded by the number of its characters (see section 5.2.3, module POUT)
SET P TO ZBOU
CALL PRINT
SETA ZX+
/LINE NUMBER
Y
ZERO
PRINT LINE NUMBER
NOLINE
UPDATE CONVERSION PARAMETER (NO ZERO SUPPRESSED)
CALL DMPR
CALL DUMP
GOTO FAIL

Note: The registers are not restored on exit

5.2.3 INTEGRATION STRUCTURE

The integrator is called to link-load the following:

1. The Job's master segment

2. The code segment: This consists of three modules

   2.1 MAIN: The code produced by the Pascal Compiler

   2.2 PRTM: The run time monitor

   2.3 BUFFER: The buffering and conversion subroutines

3. An input segment consisting of the input module PIN

4. An output segment consisting of the output module POUT

5. Another output segment consisting of the output module POUT1

6. The data segment, consisting of the module GLOBAL (data area of MAIN).

From the above modules only MAIN and GLOBAL are produced by the Pascal Compiler, the rest are 'standard'.

The input channel is channel (SC1). Default is the paper tape reader (disc).

The output channel is channel (SC2). Default is the line printer.

The other output channel is (SC3). Default is the paper tape punch (disc).

Note: The input and output modules are integrated in separate modules to allow for simultaneous I/O transfers, when the full Executive becomes available.

Module MAIN

This consists of a series of calls to the macros of section 5.1 . The series should always contain

```
PRUB=M=SIZE (program size) )
PANS=MAIN                  ) main program
PENT=0=Param2=Param3=T=0   )   entry
```

```
and          PEXT=2=Param2=F=Param4 (any)/MAIN exit ,

and end with PRUB=E=SIZE
```

Note: 2 words are reserved for the return address and the dynamic pointer for the main program.

Module PRTM

See section 5.2.2, 5.2.3.

Module Buffer.

See [2] .

Module PIN

This consists of the following:

a) A six word cell area.
Any data needed for input could be inserted here.

b) INPT=IN=[0086]=(SC1)=41.
See[2]. Six-character signed numbers are assumed. Separator any.

c) The input buffer. This is 41 words long; 80 characters plus CR, LF.

Module POUT

This consists of the following:

a) A six word cell area

b) Data for the PRINT, DMPR and DUMP subroutines

c) /

c) OUTP=OU=[0002]=(SC2)=66

See [2].

d) The output buffer. This is 66 words long, i.e. 132 characters.

e) A series of messages. Each message is preceded by the number of characters in it.

See PCHK macro section 5.1.9 .

Module POUT1

This should contain:

a) A six word cell area

b) A boolean (TRAIL) to indicate if trailer has been produced or not (paper tape punch)

c) OUTP=OW=[0046]=(SC3)=66

d) An output buffer of 66 words long.

Module GLOBAL

This is output by the Pascal Compiler. It should start with

```
PRUB=G=SIZE
PANS=GLBL
```

which putsthe program size as the first word of the module and the label PGLBL after it.

It should then be followed by a series of PSTR macro calls (see section 5.1.5).

It should end with PRUB=F==SIZE=

REFERENCES

[1] Yardy D.T. : Specification for the ALP Type 2.
ICS Ltd., 1972.

[2] ? : Buffering and Conversion Subroutines.
ICS Ltd.

[3] ? : Integrator Parameter Language (IPL)
Ancillary Specification.
ICS Ltd.

[4] Ashurst C.A. : Symbolic Usercode Language (SUL)
ICS Ltd., 1971.

[5] Gries D. : Compiler Construction for Digital Computers.
John Wiley & Sons, Inc. 1971. Chapters 17, 18.